4

ISI Research Report
ISI/RR-88-228
December 1988

University
of Southern
California

Johanna D. Moore
William R. Swartout

Explanation in Expert Systems: A Survey

INFORMATION
SCIENCES
INSTITUTE
ISI
213/822-1511
4676 Admiralty Way/Marina del Rey/California 90292-6695

Unclassified
SECURITY CLASSIFICATION OF THIS PAGE

ADA206283

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION
Unclassified

1b. RESTRICTIVE MARKINGS

2a. SECURITY CLASSIFICATION AUTHORITY

2b. DECLASSIFICATION / DOWNGRADING SCHEDULE

3. DISTRIBUTION / AVAILABILITY OF REPORT
This document is approved for public release; distribution is unlimited.

4. PERFORMING ORGANIZATION REPORT NUMBER(S)
ISI/RR-88-228

5. MONITORING ORGANIZATION REPORT NUMBER(S)

6a. NAME OF PERFORMING ORGANIZATION
USC/Information Sciences Institute

6b. OFFICE SYMBOL *(If applicable)*

7a. NAME OF MONITORING ORGANIZATION

6c. ADDRESS *(City, State, and ZIP Code)*
4676 Admiralty Way
Marina del Rey, CA 90292-6695

7b. ADDRESS *(City, State, and ZIP Code)*

8a. NAME OF FUNDING / SPONSORING ORGANIZATION Defense Advanced Research Projects Agency

8b. OFFICE SYMBOL *(If applicable)*

9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
NASA-Ames: NCC 2-520

8c. ADDRESS *(City, State, and ZIP Code)*
1500 Wilson Boulevard
Arlington, VA 22209

10. SOURCE OF FUNDING NUMBERS

PROGRAM ELEMENT NO.	PROJECT NO.	TASK NO.	WORK UNIT ACCESSION NO.
-------	-------	-------	-------

11 TITLE *(Include Security Classification)*
Explanation in Expert Systems: A Survey (Unclassified)

12. PERSONAL AUTHOR(S) Moore, Johanna D.; Swartout, William R.

13a. TYPE OF REPORT
Research Report

13b. TIME COVERED
FROM ________ TO ________

14. DATE OF REPORT *(Year, Month, Day)*
1988, December

15. PAGE COUNT
59

16. SUPPLEMENTARY NOTATION

17. COSATI CODES

FIELD	GROUP	SUB-GROUP
09	02	

18. SUBJECT TERMS *(Continue on reverse if necessary and identify by block number)*
expert systems, expert systems explanation, natural language generation, question-answering processing

19. ABSTRACT *(Continue on reverse if necessary and identify by block number)*

In order to be considered useful and acceptable, expert systems must be able to explain their knowledge of the domain and the reasoning process they employ to produce results and recommendations. Despite the fact that the need for explanation has been widely recognized, current expert systems have only limited explanatory capabilities. In this survey, we review early approaches to explanation in expert systems and discuss their limitations. We discuss improvements to the explanation capabilities based on enriched knowledge bases of expert systems. We then argue that further improvements in explanation require better generation techniques. Related work in the field of natural language generation suggests techniques that are useful to the task of explanation in expert systems; however, even those techniques will not provide all of the capabilities required for the task of carrying on a dialogue with the user. Finally, we describe our approach to explanation, which provides the facilities necessary to carry on an interactive dialogue with the user.

20. DISTRIBUTION / AVAILABILITY OF ABSTRACT
☒ UNCLASSIFIED/UNLIMITED ☒ SAME AS RPT. □ DTIC USERS

21. ABSTRACT SECURITY CLASSIFICATION
Unclassified

22a. NAME OF RESPONSIBLE INDIVIDUAL
Victor Brown Sheila Coyazo

22b. TELEPHONE *(Include Area Code)*
213/822-1511

22c. OFFICE SYMBOL

83 APR edition may be used until exhausted.
All other editions are obsolete.

SECURITY CLASSIFICATION OF THIS PAGE
Unclassified

ISI Research Report
ISI/RR-88-228
December 1988

University of Southern California

Johanna D. Moore
William R. Swartout

Explanation in Expert Systems: A Survey

INFORMATION SCIENCES INSTITUTE

213/822-1511
4676 Admiralty Way/Marina del Rey/California 90292-6695

This research is supported by the Defense Advanced Research Projects Agency under NASA-Ames Cooperative Agreement NCC 2-2520. Views and conclusions contained in this report are the authors' and should not be interpreted as representing the official opinion or policy of DARPA, NASA, the U.S. Government, or any person or agency connected with them.

Explanation in Expert Systems: A Survey

Johanna D. Moore
UCLA Department of Computer Science
and
USC/Information Sciences Institute

William R. Swartout
USC/Information Sciences Institute

Abstract

In order to be considered useful and acceptable, expert systems must be able to explain their knowledge of the domain and the reasoning processes they employ to produce results and recommendations. Despite the fact that the need for explanation has been widely recognized, current expert systems have only limited explanatory capabilities. In this survey, we review early approaches to explanation in expert systems and discuss their limitations. We discuss improvements to the explanation capabilities based on enriched knowledge bases of expert systems. We then argue that further improvements in explanation require better generation techniques. Related work in the field of natural language generation suggests techniques that are useful to the task of explanation in expert systems; however, even those techniques will not provide all of the capabilities required for the task of carrying on a dialogue with the user. Finally, we describe our approach to explanation, which provides the facilities necessary to carry on an interactive dialogue with the user.

1 Introduction

In order to be considered useful and acceptable, expert systems must be able to explain their knowledge of the domain and the reasoning processes they employ to produce results and recommendations. Expert systems researchers have identified several reasons why explanation capabilities are not only desirable, but crucial to the success of expert systems. These include assisting both users and system builders in understanding the contents of the system's knowledge base and reasoning processes; facilitating the debugging of the system during the development stages; educating users both about the domain and the capabilities of the system; and persuading users that the system's conclusions are correct so that they can ultimately accept these conclusions and trust the system's reasoning powers [BS84]. In addition, an explanation facility can be useful for assessing the system's appropriateness for a given task. The scope of an expert system may be quite narrow and explanation can help a user discover when a system is being pushed beyond the limits of its knowledge [Swa83].

The critical need for explanation has been voiced not only by expert system builders, but by the intended user community as well. When physicians were asked to rank 15 capabilities of computer-based consultation systems in order of importance, they ranked the ability "to explain their diagnostic and treatment decisions to physician users" as the most essential of the capabilities surveyed [TS84]. Third on the list was the ability to "display an understanding of their own medical knowledge." The desirability of explanation capabilities to these users is underscored by the fact that the capability to "never make an incorrect diagnosis" was ranked 14th out of 15 capabilities surveyed!

Despite the fact that the need for explanation has been widely recognized, current expert systems have only limited explanatory capability. In particular, the explanation facilities of most current systems can be characterized as:

- **narrow**: only a few types of questions can be answered
- **inflexible**: explanations can be presented in only one way
- **insensitive**: explanations cannot be tailored to meet the needs of different users or of different situations
- **unresponsive**: the system cannot answer follow-up questions or offer an alternative explanation if a user does not understand a given explanation
- **inextensible**: new explanation strategies cannot be added easily.

These problems stem from limitations in two areas. First, the knowledge bases of current systems are inadequate in many ways. Until recently, expert systems architects have concentrated their efforts on the problem-solving needs of the system. They have designed knowledge bases and control mechanisms best suited to performing the expert task. Because of this, the explanation capabilities of these systems are limited to producing procedural descriptions of *how* a given problem is solved. Knowledge needed to justify the system's actions, explain general problem solving strategies, or define the terminology used by the system is simply *not* represented and theretore cannot be included in explanations [Cla83b], [Swa83]. Deficiencies in the knowledge representation contribute to the inflexibility, insensitivity, and unresponsiveness of these systems as well. To provide different explanations to different users, in different situations, or in response to requests for elaboration, the system must have a rich representation of its knowledge. The system must have abstract strategic knowledge as well as detailed knowledge, a rich terminological base, and causal knowledge of the domain.

But only half of the problem can be solved by improvements in the knowledge base. Once additional knowledge is made available, that knowledge must be employed by the explanation mechanism. The second area of weakness in current systems lies in their text planning and generation strategies. Expert system builders, wishing to circumvent the difficult problems in natural language generation and discourse, have attempted to provide explanations by exploiting simple strategies such as translating the sequence of steps the system executed in producing a result, or filling in the blanks in templates. Much of the text produced by these systems is canned into the translation rules or templates. The explanation strategies are extremely rigid; there are few choices to be made after selecting the appropriate template. While these systems are capable of producing multi-sentential texts, they have little, if any, understanding of the text they produce and have no model of how the clauses in the text relate to one another or what role individual clauses in the text play in responding to the user's query.

Moreover, such strategies assume that a trace of the procedures followed by the program in solving a problem will produce an explanation that is coherent to end users. But, as other researchers ([Dav76], [WJ82], [Swa83], [Cla83b], [PC84]) have noted, and the examples in the next section illustrate, the computationally efficient reasoning strategies used by such programs to produce a result often *do not* form a good basis for understandable explanations. This suggests that this assumption be abandoned, and that we come to regard explanation production as a *problem-solving activity* in its own right. In this view, the explainer is confronted with the problem of determining the most appropriate way to convey information to the user. In forming an explanation, the explainer may employ the knowledge captured in the system's reasoning methods and execution trace, but it is not limited to it. Thus, we are advocating a separation of the process of explanation production from that of domain problem solving so that the structure of the explanations no longer slavishly follows that of the reasoning methods or execution trace. As Webber suggests, the explainer should *use* the program's reasoning to "suggest and instantiate conceptually more accessible strategies for organizing and presenting justifications" [WJ82]. Davis argues that an approach in which problem solving and explanation are distinct more accurately reflects human behavior. An expert's account of how he solved a complex problem may be quite different from a simple recap of his actions because the process of solving the problem often produces new insights or views of the problem. These may lead to a "more compact explanation" that cannot be mimicked by simply omitting details [Dav76].

Existing explanation strategies have proven inadequate for current systems and will become even less viable as knowledge bases become more sophisticated. If we are to address the problems cited above, explanation must be treated as what it is: a problem in text-planning and natural language generation. Instead of simply translating code into English, an explanation facility must employ sophisticated text-planning strategies of the kind developed by researchers studying natural language generation. Explainers of the future must have strategies that are based on those observed in naturally-occurring text. These strategies must be flexible so that explanations can be tailored to the needs of individual users and specific dialogue settings. Explainers must have alternative strategies for producing responses so that they may provide elaboration or clarification when users are not satisfied with the first explanation given. Furthermore, the system must be able to interpret follow-up questions about misunderstood explanations or requests for elaboration in the context of the dialogue that has already occurred, and not as independent questions.

Although an enhanced knowledge base is necessary to support such strategies, a more sophisticated knowledge base will actually make the task of designing explanation strategies more difficult since the explainer will now have to make many more choices when producing an explanation. For example, if the system represents its knowledge at various levels of abstraction, the explainer must choose a level of abstraction for presenting its explanations. It must decide which steps to elaborate and whether to use abstract or specific terminology. Future explanation mechanisms will need an understanding of how explanations should be structured so that they will be able to choose among the many alternatives.

In this paper, we describe early approaches to explanation generation in expert systems, analyzing two systems as case examples and discussing their limitations in detail. We then briefly discuss two approaches that confront the problem of capturing more of the knowledge needed to support sophisticated explanation capabilities. To confront the problem of devising sophisticated explanation techniques, we draw on the natural language generation literature. Finally, we discuss some current research in providing an explanation facility that is able to react to feedback from the user by answering follow-up questions and offering further explanation when the user indicates dissatisfaction with a given explanation.

2 Early Approaches to Explanation in Expert Systems

Early attempts to provide programs with an ability to explain their behavior fall into two categories: 1) those that produce explanations from text prepared *a priori* and associated with structures or procedures in the program, and 2) those that produce explanations by translating directly from the program code and execution traces.

2.1 Explanation From Canned Text or Templates

In the first approach, system builders anticipate what questions are likely to be asked and construct the natural language text of the responses to be given. This text is often referred to as *canned text* and explanations produced by displaying this text are *canned explanations.* The canned text may contain variables to be filled in with values from a specific execution of the program. Structures that mix canned text with slots to be filled in are called *templates.* Error messages in programs are a common example of this approach.

The advantage of this approach is its simplicity. Once a program is written, text is prepared and associated with each part of the program to be explained. When a user requests explanation of some aspect of the program's behavior, the text associated with that portion of the program is simply displayed.

However, the canned text approach has several serious drawbacks. First, since the text strings used to explain the code are independent from the code itself, inconsistency between the code and the text strings can arise as the code is modified. This makes it more difficult to be certain that the system is actually doing what it claims to be doing. In addition, since the system can only answer questions it has appropriate text strings for, all questions the system can answer must be anticipated in advance and the appropriate explanation templates added to the system. This may be feasible for small systems expected to answer a very limited set of questions, but for large systems it will become increasingly difficult to anticipate all of the questions that users would like to ask. In addition, each time new functionality is added to the code, templates to explain that functionality must be added and their interaction with existing templates must be understood.

The most serious drawback of the canned text approach is that the system has no conceptual model of the explanations it is producing. It has no representation of the purpose an explanation is serving or how parts of the explanation are related to one another in achieving that purpose. To an explanation system of this type, all explanations are simply strings of characters associated with certain parts of the program.

2.2 Explanation By Translating the Code

In an attempt to alleviate the problems of inconsistency and inextensibility associated with the canned text approach, researchers developed explanation systems which produce their explanations directly from the system's code. In this approach, programs are written in a language to which simple transformations can be applied to produce natural language explanations of the code's behavior.

To illustrate this approach, we will consider two systems in detail. The first is the Digitalis Advisor [GSP78], [Swa77], which used a translate-the-code approach to generate explanations of the system's behavior. The second is the original MYCIN system [Sho76], [Dav76], [BS84], which produced its explanations by combining the translate-the-code and canned text approaches.

2.2.1 The Digitalis Advisor

The Digitalis Advisor is a program designed to advise physicians regarding digitalis therapy. In structuring the Digitalis Advisor, an attempt was made to model the structure of the expert's problem-solving techniques. The program is structured into *levels of abstraction*, where higher-level procedures are intended to represent more general goals or actions. High-level procedures call more specific procedures, which in turn call still more specific procedures, and so on. For example, the procedure used to begin treating a patient is BEGIN THERAPY. One of the procedures that this procedure calls is CHECK SENSITIVITIES, which checks for any sensitivities the patient may have to the drug. CHECK SENSITIVITIES in turn calls a number of subprocedures, one for each type of sensitivity that must be considered. One of these is CHECK SENSITIVITY DUE TO THYROID FUNCTION.

When the system is asked to describe how therapy begins, the call to CHECK SENSITIVITIES will be mentioned as a step in the process, but will not be further elaborated. If the user wishes to know how sensitivities are checked, he may ask the system to describe CHECK SENSITIVITIES and the system will then give the details of this procedure, including steps such as CHECK SENSITIVITY DUE TO THYROID FUNCTION. A user who wishes to know the details of how this particular sensitivity is checked may inquire further. Figure 1 shows the explanation generated by the Digitalis Advisor in response to such a query.[1]

There are some advantages to the translate-the-code approach. The explanation strategies employed by the Digitalis Advisor are extremely simple. The structure of the explanations follows the structure of the program code exactly and the names of variables in the explanation are those used in the program. For this reason, explanations produced in this manner are extremely useful for system builders developing and debugging the system. In addition, because the explanations are produced directly from the code, changes in the code will be reflected in the system's explanations automatically. Thus, consistency between code and explanation is guaranteed.

But there are also limitations to this approach. We will return to these in Section 3 after we have discussed the MYCIN system.

2.2.2 MYCIN

MYCIN is a rule-based medical consultation system designed to provide advice regarding diagnosis and therapy for infectious diseases [Sho76], [Dav76], [BS84]. MYCIN's medical

[1] This example is actually medically incorrect – myxedema is *not* a sensitivity. This error in the Digitalis Advisor was detected only when the system was recoded in the XPLAIN framework to provide better explanations!

```
To check sensitivity due to thyroid-function I do the following steps:

1. If the current value of the status of myxedema is unknown then
   I ask the user the level of T4.

2. I do one of the following:

   2.1 If either the status of myxedema is present or the status of
       myxedema is unknown and the level of T4 is less than 2.50
       then I do the following substeps:

       2.1.1 I add myxedema to the present and correctable conditions.
       2.1.2 I remove myxedema from the degradable conditions.
       2.1.3 I set the factor of reduction due to myxedema to 0.67.
       2.1.4 I add myxedema to the reasons of reduction.

   2.2 Otherwise, I add myxedema to the degradable conditions,
       remove myxedema from the present and correctable
       conditions, set the factor of reduction due to myxedema
       to 1.00 and remove myxedema from the reasons of reduction.
```

Figure 1: Explaining How A Task Is Accomplished

knowledge is encoded in a set of rules. Each rule is intended to represent a single, independent "chunk" of domain-specific knowledge indicating a conclusion that can be drawn if the conditions specified in the premise are satisfied. Figure 2 shows the internal representation of a rule with its English translation. Rules are composed from a small set of primitive functions that make up the rule language. Associated with each of the primitive functions (e.g., AND, SAME, MEMBF) is a template with blanks to be filled in by translations of the function's arguments (e.g., CNTXT, SITE, GI). These templates are used when generating explanations.

A consultation is run by backward chaining through applicable rules, asking questions of the user when necessary. For example, if the program is attempting to determine the identity of an infecting organism (i.e., determining the identity of the organism is the system's *goal*), it retrieves all of the rules that draw a conclusion about the identity of the organism. It then invokes each rule in turn by evaluating the rule's premises to see whether or not all of the necessary conditions are satisfied. For example, when the rule in Figure 2 is invoked, evaluating the first premise requires determining the type of the infection. This now becomes a *subgoal* and the process of finding applicable rules and invoking them recurs. As the consultation progresses, MYCIN builds a *history tree* reflecting the goal/subgoal structure of the executing program.

Explanations are produced from the history tree using the rule templates to translate the sequence of rules that were applied to reach a conclusion. While the consultation is in progress, whenever MYCIN asks a question, the user may examine the current reasoning

```
PREMISE: (AND (SAME CNTXT INFECT PRIMARY-BACTEREMIA)
              (MEMBF CNTXT SITE STERILESITES)
              (SAME CNTXT PORTAL GI))
ACTION: (CONCLUDE CNTXT IDENTITY BACTEROIDES TALLY .7)
```

IF: 1) The infection is primary-bacteremia, and
   2) the site of the culture is one of the sterile sites, and
   3) the suspected portal of entry of the organism is the gastro-intestinal tract,

THEN: There is suggestive evidence (.7) that identity of the organism is bacteroides.

Figure 2: Internal Representation and English Translation of MYCIN Rule

chain by asking why the system is asking that question. To answer such questions, the system looks up in the history tree to see what higher-level goal it is attempting to satisfy. Limited forms of follow-up questions are handled. Repeatedly asking "why" allows the user to ask about successively higher-level goals. For any of the subgoals mentioned in the answer to a why-question, the user may ask how this goal was (will be) achieved. To answer how-questions, the system looks down in the history tree to find all the ways that a subgoal was (may be) achieved. This may involve looking at the execution trace to see what rules actually contributed to the conclusion. An example from [BS84] demonstrating MYCIN's explanation capabilities, including repeated why and how questions, appears in Figure 3.

Users may also ask questions once the consultation is complete. MYCIN responds to questions about the current state of the system's knowledge base, including both static domain knowledge and facts accumulated during a particular consultation. Questions are parsed using a keyword approach, relying strongly on the very specific vocabulary of the domain to understand what is being asked. Questions are classified into a limited set of question types. Corresponding to each question type, there are a number of possible answer templates, one of which will be chosen based on the history tree and the system's knowledge base.

These templates are very rigid in structure. For example, for questions of the form "How do you know the value of <parameter> of <context>?" the two answer templates are:

```
I used <rule> to conclude that <parameter> of <context> is <value>.
This gave a cumulative Certainty Factor of <certainty factor>.
The last question asked before the conclusion was made was <question
number>.
```

```
15) Is the patient's illness with ORGANISM-1 a hospital acquired infection?
**WHY?
[i.e., WHY is it important to determine whether or not the infection with
     ORGANISM-1 was acquired while the patient was hospitalized?]
[1.0] This will aid in determining the category of ORGANISM-1.
          It has already been established that
                [1.1] the gram stain of ORGANISM-1 is gramneg, and
                [1.2] the morphology of ORGANISM-1 is rod, and
                [1.3] the aerobicity of ORGANISM-1 is facultative
          Therefore, if
                [1.4] the infection with ORGANISM-1 was not acquired while the
                      patient was hospitalized
          then there is strongly suggestive evidence (.8) that the category of
                      ORGANISM-1 is enterobacteriaceae
          ALSO: there is weakly suggestive evidence (.1) that the identity
                      ORGANISM-1 is pseudomonas-aeruginosa
          [RULE037].
**WHY?
[i.e., WHY is it important to determine the category of ORGANISM-1?]
[2.0] ... in order to determine the identity of ORGANISM-1.
          It has already been established that
                [2.1] this blood culture was taken from a sterile source
          Therefore, if
                [2.2] this current organism and at least one of the list of members
                      associated with the category of the organism agree with
                      respect to the following properties: air conformation
          then there is strongly suggestive evidence (.9) that each of them is the
                      identity of ORGANISM-1
          [RULE003].
**HOW 1.3?
[i.e., HOW was it established that the aerobicity of ORGANISM-1 is facultative?
The following were used:
[3.1] RULE027 indicated there is strongly suggestive evidence (.8) that the
     aerobicity of ORGANISM-1 is facultative
[2.1] RULE027 indicated there is weakly suggestive evidence (.2) that the
     aerobicity of ORGANISM-1 is anaerobic
Since this gave a cumulative CF of (.8) for facultative, and (.2) for
      anaerobic, it has been established that the aerobicity of ORGANISM-1
      is facultative.
**HOW 1.1?
[i.e., HOW was it established that the gram stain of ORGANISM-1
     is gramneg?
You said so [question 11].
```

Figure 3: A Sample MYCIN Explanation

```
In answer to question <question number> you said that
<parameter> of <context> is <value>.
```

So we see that MYCIN uses a combination of the translate-the-code and canned text approaches. To answer questions about what the program is doing, MYCIN follows the history tree translating the rules that have been executed. To answer other types of questions, MYCIN relies on templates containing much canned text. As in the Digitalis Advisor, the advantage of MYCIN's approach is that the explanation strategies are simple. However, both MYCIN and the Digitalis Advisor have many limitations, which we now discuss.

3 Limitations of the Early Approaches to Explanation

Because the early approaches to explanation were so simple and limited, there are many types of questions users would like to ask that these systems are not capable of answering or to which the responses generated are unsatisfactory. In a study of several examples of inadequate explanations that MYCIN generated in response to questions asked by users [BS84], the implementors determined that problems arose because of: (1) MYCIN's lack of support knowledge, i.e. the underlying mechanistic or associational links that explain why the action portion of a rule follows from its premises; (2) MYCIN's failure to deal with the context in which a question was asked - MYCIN has no sense of dialogue, so each question requires full specification of the points of interest without reference to earlier exchanges; and (3) a misinterpretation of the user's intent in asking a question. The study identified examples of simple questions that could mean four or five different things depending on what the user knows, the information currently available about the patient under consideration, or the content of the earlier discussions.

These limitations can be broken down into two general problems: 1) the lack of knowledge needed to support the production of explanations, and 2) the lack of a general model of explanation.

3.1 Impoverished Knowledge Bases

A severe limitation of systems that generate explanations solely by translating their code is that these systems *cannot* give justifications for their actions. These systems can state *what* they did but cannot tell the user *why* they did it or why they did things *in the order they did them.* For example, in the explanation produced by the Digitalis Advisor in Figure 1, the system cannot give a causal rationale for reducing the dose of digitalis if myxedema is present.

In attempting to adapt MYCIN for use as a tutoring system, Clancey examined MYCIN's rule base and found that individual rules serve different purposes, have different justifications, and are constructed using different rationales for the ordering of clauses in their premises [Cla83b]. However, these purposes, justifications and rationales are not explicitly included in the rules; therefore many types of explanations simply are not possible. For example, consider the rule shown in Figure 4 and suppose that the user wishes to know why the five clauses in its premise suggest that the organism causing the infection may be diplococcus or e.coli. MYCIN cannot explain this because the system knows no more about the association between the premises and conclusion than what is stated in this rule.

IF: 1) the infection which requires therapy is meningitis,
2) only circumstantial evidence is available for this case,
3) the type of meningitis is bacterial,
4) the age of the patient is greater than 17 years old, and
5) the patient is an alcoholic,

THEN: there is evidence that the organisms which might be causing the infection are diplococcus-pneumoniae (.3) or e.coli (.2)

Figure 4: A MYCIN Rule With Implicit Knowledge

In particular MYCIN does *not* know that clauses 1, 3, and 5 together embody the causal knowledge that if an alcoholic has bacterial meningitis, it is likely to be caused by diplococcus.[2] Furthermore, MYCIN does *not* realize that clause 4 is a *screening clause* that prevents the system from asking whether the patient is an alcoholic when the patient is not an adult – thus making it appear that MYCIN understands this social "fact." However, MYCIN does not explicitly represent, and therefore cannot explain, this relationship between clauses 4 and 5. Even worse, not knowing that the system makes this assumption may lead the user to infer that age has something to do with the type of organism causing the infection.

Another hidden relationship exists between clauses 1 and 3. Clearly bacterial meningitis is a type of meningitis, so why include clause 1? The ordering of clauses 1 and 3 implicitly encodes strategic knowledge about how the program carries out a consultation. The justification for the order in which goals are pursued is implicit in the ordering of the premises in a rule. The choice of ordering for the premises is left to the discretion of the rule author and there is no mechanism by which he can record the rationale for his choices. This makes it impossible for MYCIN to explain its general problem-solving strategy, i.e. that it establishes that the infection is meningitis before it determines if it is bacterial meningitis because it is following a refinement strategy of diagnosing the disease [HCR84].

Furthermore, the order in which rules are tried to satisfy a particular goal will affect the order in which subgoals are pursued. Recall that when a goal such as determining the identity of the organism is being pursued, MYCIN invokes all of the rules that conclude about the identity of the organism in the order in which they appear in the rule base. This order is determined by the order in which the rules were entered into the system! Thus MYCIN cannot explain why it considers one hypothesis before another in pursuing a goal. In addition, because attempting to satisfy the premises of a rule frequently causes questions to be asked of the user, MYCIN cannot explain why it asks questions in the order it does because this too depends on the ordering of premises in a rule and the ordering of rules

[2] Diplococcus is normally found in the mouth and throat. The fact that the patient is an alcoholic allows access of the organisms from the throat and mouth to the lungs by reaspiration of secretions. The organism passes from the lungs to the meninges by the blood. The organism finds favorable conditions for growth because the alcoholic patient is a compromised host and is therefore susceptible to infection.

in the knowledge base. As Clancey points out in [Cla83b], "focusing on a hypothesis and choosing a question to confirm a hypothesis are not arbitrary in human reasoning" and thus users will expect the system to be able to explain why it pursues one hypothesis before another and will expect questions to follow some explicable line of reasoning.

Another type of knowledge that is not explicitly represented in these systems is terminological knowledge. For example, one of the steps in the Digitalis Advisor's procedure BEGIN THERAPY is CHECK SENSITIVITIES, but what *is* a sensitivity and what do sensitivities have to do with digitalis therapy? Users who are novices in the task domain will need to ask questions about terminology to understand the system's responses and to be able to respond appropriately when answering the system's questions. Experts may want to ask such questions to determine whether the system's use of a term is the same as their own. Because the knowledge of what a term means is implicit in the way it is used in the rules or procedures that make up the the system's knowledge base, the system is not capable of explaining what such terms *mean* in a way that is acceptable to users. An effort to explain terms by examining the rule base of an expert system [Rub85] has been only partially successful because so many different types of knowledge are encoded into the single, uniform rule formalism. This makes it difficult to distinguish definitional knowledge from other types of knowledge.

As we have seen, rules and rule clauses incorporate many different types of knowledge, but the uniformity of the rule representation obscures their various functions thus making comprehensible explanation impossible. Much of the information that went into writing the rules (in the case of MYCIN) or the program (in the case of the Digitalis Advisor) including justification and strategic information is either lost completely or made implicit in the rules or program code therefore is not available to be explained. Both Swartout [Swa81] and Clancey [Cla83b] have argued that the different types of knowledge (definitional, world facts, causal, strategic) must be separately and explicitly represented if systems are to be able to explain their strategy and justify their conclusions. As a consequence, later efforts have addressed the problem of capturing the knowledge and decisions that went into writing the program and explicitly representing this information so that it will be available for explanation. This research is briefly discussed in Section 4 For a more detailed discussion of knowledge representation issues and a description of the Explainable Expert Systems approach, see [SS87b], [SS87a].

3.2 Lack of General Model of Explanation

3.2.1 The "Recap as Explanation" Myth

The translate-the-code approach places much of the burden of producing explanations on the programmer. It relies on the programmer's ability to structure the program in a way that will be understandable to users who are knowledgeable about the task domain. For example, because the Digitalis Advisor is hierarchically structured into procedures that model the expert's solution to the digitalis therapy problem, the explainer effectively summarizes a procedure by mentioning the call to that procedure. This summarization is an artifact of the inherent structure of problem solving in the domain of digitalis therapy and the way the program has been designed. It is *not* attributable to any sophistication in the explanation strategies. Similarly, in MYCIN, an attempt was made to make each rule an independent "chunk" of medical knowledge so that stating a single rule would be a complete, coherent explanation.

Unfortunately, program structure is also dictated by implementation considerations which may obscure the underlying domain-related reasoning. Many operations that are explicit in the computational procedures would be implicit in the minds of the domain experts as they solve the problem. For example, in the sample explanation in Figure 1, steps 2.1.1, 2.1.2, 2.1.4, and much of 2.2 refer to the management of variables that keep track of which sensitivities have been checked and the reasons for dosage reduction. These steps have much more to do with the internal idiosyncrasies of the program than with medical reasoning, but they appear in the code and so are described by the explanation routines.

Moreover, because the explanation is cluttered with implementation details, it may be more confusing than enlightening to physician-users because it is very difficult for a user to get an idea of what is really going on. Consider again the example in Figure 1. While the program is excrutiatingly explicit about its internal bookkeeping, it is not very explicit about the fact that it is trying to decide whether or not to reduce the dose of digitalis depending on the status of myxedema and the level of T4. This is only opaquely hinted at by the phrases *"I set the factor of reduction due to myxedema to 0.67"* in one case and *"I set the factor of reduction due to myxedema to 1.00"* in the other. Referring to no reduction in the dosage as "setting the factor of reduction to 1.00" may make perfect sense to programmers, but it is certainly not the way domain experts would express this notion.

3.2.2 Inadequate Natural Language Techniques

The problems discussed in the introduction and identified by the MYCIN implementors in their analysis of inadequate responses also stem from limitations in the natural language capabilities of current systems. First, question understanding and interpretation procedures are limited, thus restricting the kinds of questions that may be asked and the manner in which they must be phrased. To avoid the difficult problems of inferring the user's intent in asking a question, MYCIN interprets a user's "Why?" query in only one way even though it could have a variety of meanings, such as "Why is it important to determine....?," "Why did you ask about that instead of....?," "Why do you ask that now?," or "Why does the conclusion follow from the premises?" All of these interpretations are valid questions about the system's knowledge and behavior, yet MYCIN always assumes the first interpretation and does not allow the other questions to be asked. In the sample MYCIN explanation in Figure 3, when the user asks "Why?" the second time, the system assumes that the user is asking *"Why is it important to determine the category of ORGANISM-1?"*. But, the user may really be asking a very different type of question, namely *"Why is it the case that a gram negative, facultative rod not acquired in a hospital setting is likely to be enterobacteriaceae?"*[3] We have already seen that the causal knowledge needed to answer this question is not represented in MYCIN. But even if it were, MYCIN could not determine what "why?" question was being asked in a given context because MYCIN does not view the explanation session as an on-going dialogue. Each question-answer pair is viewed independently and references to previous portions of the dialogue can be made only in stilted and artificial ways, as shown in Figure 3. A model of explanation that addresses this problem is proposed

[3] This example is adapted from an example found in [Dav76].

in Section 6.

Another drawback is that current systems have no mechanism by which to take the user's feedback into account, i.e., they do not have a means for allowing the user to indicate that he is not satisfied with a given explanation. They do not explicitly represent user's goals in asking a question or the explanation strategies used to achieve them. These systems do not know what implicit assumptions must hold true for the explanation to be understood by the user or what dependencies exist between the various parts of the text produced. In short, current systems do not really have an understanding of the explanations they generate. This limitation makes repairing misunderstood explanations impossible.

Moreover, current systems usually have only a single presentation strategy associated with each question type, instead of the sophisticated repertoire of discourse strategies that human explainers utilize. Thus, these systems could not provide alternative or additional explanations even if they did allow the user to indicate that he would like elaboration or clarification.

Due to these limitations, users must phrase their questions in a form acceptable to the system and have little recourse if they do not understand or are not satisfied with an explanation as it is presented to them by the system.

In the next two sections, we discuss approaches to alleviate some of the limitations we have identified. First we discuss efforts to explicitly capture the knowledge that is needed to produce explanations and then we present more sophisticated natural language techniques for conveying this knowledge to the user.

4 Capturing More Knowledge with Improved Architectures

We have seen that early systems were not able to give abstract explanations of their problem-solving strategies or to justify their behavior. Researchers realized that in order to achieve these capabilities, their systems would need to represent strategic knowledge explicitly rather than leaving it implicitly embedded in the domain knowledge, such as in the ordering of premises in a rule or the ordering of rules in the knowledge base. Two systems, NEOMYCIN [Cla83a] and XPLAIN [Swa83], have taken the approach of representing strategic knowledge explicitly and separately from domain knowledge. These systems are able to produce both abstract and concrete explanations of their reasoning strategies and the XPLAIN system is able to justify its results in terms of the causal model of the domain.

4.1 NEOMYCIN

NEOMYCIN [CL81] arose out of Clancey's frustration in attempting to adapt MYCIN to a system that could be used for educational purposes [CSB84], [Cla83b]. Clancey noted that in order to teach medical students about diagnosis, it was necessary to be able to justify the diagnostic associations encoded in MYCIN's rules and to explain the overall diagnostic strategy of gathering information and focusing on hypotheses. As discussed in the preceding section, this knowledge was not explicitly represented in the MYCIN knowledge base and therefore could not be explained.

In NEOMYCIN, a domain-independent diagnostic strategy is represented explicitly and separately from knowledge about the domain – the disease taxonomy, causal and

METARULE411
IF: the datum in question is strongly associated with the current focus
THEN: apply the related list of rules
TRANS: ((VAR ASKINGPARM) (DOMAINWORD “triggers) (VAR CURFOCUS))

METARULE566
IF: the datum in question makes the current focus more likely
THEN: apply the related list of rules
TRANS: ((VAR ASKINGPARM) “makes” (VAR CURFOCUS)) “more likely”

Figure 5: Two Metarules for the task TEST-HYPOTHESIS

data/hypothesis rules, and world facts. Knowledge of the diagnostic strategy is represented as a set of *tasks*, which are meta-level goals, and *metarules* for achieving these goals. An ordered collection of metarules defines a *generic procedure* for achieving a task. Each metarule has a premise indicating when the metarule can be applied, and an action that tells what should be done when the premise is satisfied. The action clauses of metarules determine what task should be done next, what domain rule applied, or what domain finding should be requested from the user. (See [Cla83a] for a detailed discussion of NEOMYCIN and the organization of its knowledge base.) Figure 5 shows two of the metarules that are part of the procedure for the task TEST-HYPOTHESIS.

NEOMYCIN can provide abstract as well as concrete explanations. A concrete explanation of problem-solving strategy refers to aspects of the solution to a particular problem while an abstract explanation articulates a general principle that can be applied in many different situations. NEOMYCIN's explanations are generated at the level of general strategies, instantiated with domain knowledge when possible, to make them more concrete. To explain why NEOMYCIN is doing a particular task, the question is first restated as *“Why are we trying to do this task?”* Since each task is invoked by a metarule that is part of a method for achieving some higher-level task, a suitable response may be generated by citing this higher-level task and the metarule being applied. An example of an explanation produced using this strategy appears in Figure 6. In this example, the current task is to apply domain rules. The metarule that invoked this task is METARULE566 shown in Figure 5. In the process of applying one of the rules, the user is asked a question. The higher-level task that this metarule is trying to achieve is TEST-HYPOTHESIS and the current hypothesis is that Mary has tuberculosis. Figure 6 shows an example of providing a concrete explanation. The user may also request an abstract explanation, as shown in Figure 7.

As with MYCIN, NEOMYCIN users may also ask how a previously mentioned task was or is being accomplished. An example appears in Figure 8. The explanation strategy for answering “how” questions presents all of the metarules associated with the task that have already been completed, as well as any metarule associated with the task that is currently being executed. Note that the strategy for answering “why” questions states the premise of the metarule, i.e., the reason the metarule was applicable. The strategy for responding

```
14) Does Mary have a history of granulomas on biopsy of the liver,
    lymph node, or other organ?
**WHY?
[i.e. WHY is it important to determine whether Mary has a history
    of granulomas on biopsy of the liver, lymph node, or other organ?
[19.0] We are trying to decide whether Mary has tuberculosis.
A history of granulomas on biopsy of the liver, lymph node, or other
    organ makes tuberculosis more likely.
[back to question 14 ....]
```

Figure 6: A Concrete NEOMYCIN Explanation

```
**ABSTRACT
[i.e. WHY is it important to determine whether Mary has a history
    of granulomas on biopsy of the liver, lymph node, or other organ?
[19.0] We are trying to test a hypothesis.
The datum in question makes the hypothesis more likely.
[back to question 14 ....]
```

Figure 7: An Abstract NEOMYCIN Explanation

```
**HOW 19.0?
[i.e. How are we trying to decide whether Mary has tuberculosis?
We did the following:
[20.1] Applied a rule mentioning a causal precursor to
       tuberculosis (tb risk factor; rule525), and
[20.2] Gave thorough consideration to any recent findings.
[20.3] Applied a rule mentioning general evidence for
       tuberculosis (focal neurological signs; rule366), and
[20.4] Gave thorough consideration to any recent findings.
[20.5] Applied a rule mentioning general evidence for
       tuberculosis (compromised host; rule309), and
[20.6] Gave thorough consideration to any recent findings.
We are now:
[20.7] Applying a rule mentioning general evidence for
       tuberculosis (a history of granulomas on biopsy of the liver,
       lymph node, or other organ; rule002).
We will next:
[20.8] Give thorough consideration to any recent findings.
[back to question 14 ....]
```

Figure 8: A Sample NEOMYCIN Explanation

to "how" questions states the action of the metarule, i.e., what was done.

NEOMYCIN attempts to tailor its explanations to different types of users by using different terminology depending on the type of user requesting the explanation. Users can identify themselves as either system or domain experts. Certain general words or phrases used in the rule translations have been identified as having synonyms in the vocabulary of the domain and are marked as such. Whenever a marked phrase is encountered while explaining a strategy, the corresponding domain-specific term will be substituted when the user has identified himself as a domain expert. For example, in translating METARULE411 shown in Figure 5, the phrase "is strongly associated with" will be substituted for "triggers" if the user is a domain expert.

4.2 XPLAIN

The XPLAIN system grew out of work done on the Digitalis Advisor. Swartout [Swa81] observed that the information needed to justify a program's behavior is precisely that information which was used by the programmer to implement the program and therefore what was needed was a way to capture the knowledge and decisions that went into writing the program in the first place. To do this, Swartout used an automatic program writer to record this information and thus guarantee that the program understood its own design.

SYSTEM	Please enter the value of serum potassium:
USER	Why?
SYSTEM	The system is anticipating digitalis toxicity. Decreased serum potassium causes increased automaticity, which may cause a change to ventricular fibrillation. Increased digitalis also causes increased automaticity. Thus, if the system observes decreased serum potassium, it reduces the dose of digitalis due to decreased serum potassium. Please enter the value of serum potassium:
USER	3.7
SYSTEM	Please enter the value of serum calcium:
USER	Why?
SYSTEM	The system is anticipating digitalis toxicity. Increased serum calcium also causes increased automaticity Thus, (as with decreased serum potassium) if the system observes increased serum calcium, it reduces the dose of digitalis due to increased serum calcium. Please enter the value of serum calcium:
USER	9

Figure 9: A sample XPLAIN dialogue

Like NEOMYCIN, the XPLAIN system [Swa81] represents problem solving-knowledge explicitly and separately from domain knowledge. XPLAIN's *domain model* represents the facts of the domain. For digitalis therapy, the domain model includes knowledge about physiological states, reactions to the drug digitalis, and causal relations among them. The system's problem-solving knowledge is encoded in *domain principles* that tell the system how to use its domain model to achieve a goal such as administering the proper dosage of a drug or analyzing symptoms. Domain principles have three parts: a *goal*, a *prototype method*, and a *domain rationale.* The goal indicates what task the domain principle's method is capable of achieving. The prototype method is an abstract method that tells how to accomplish the goal. The domain rationale is a pattern that indicates at a general level the cases where the domain principle should be applied. Specific cases are found by matching the domain rationale against the domain model. This process integrates the "support" knowledge in the domain model into the development of the expert system. An automatic programmer successively refines goals into prototype methods using knowledge from the domain model and leaving behind a record of the steps taken in the development of the program and the reasons behind them.

Recording the development of the actual low-level procedures from the domain principles enables the system to generate principled responses to requests for justification. An example of XPLAIN's capabilities is shown in Figure 9. Note several things from this example. First the system is capable of explaining that it is asking about the patient's serum potassium as part of adjusting the recommended dosage, and that this is important because too high a dosage of digitalis could interact with the effects of serum potassium level to produce a dangerous condition. That is, XPLAIN can justify its request for a patient parameter both by paraphrasing the code and by justifying the parameter's significance in the abstract model of the domain. This requires an understanding of causal relationships in the domain (as represented in the domain model) and of how these causal relationships are used in problem solving (as represented in the domain principles.) Recall that the original Digitalis Advisor, which had only the knowledge necessary to perform its function (i.e., correctly administer digitalis) knew how to check serum potassium and reduce the dosage, but did not know, and therefore could not explain, *why* it was doing so.

Second, by keeping track of the last few explanations produced, XPLAIN is able to suggest analogies with previous explanations. When asked about serum calcium, the system gives a shorter explanation because it knows that it has already explained several of the causal relationships in the previous explanation of serum potassium. Since the method for reducing the dose due to serum calcium and the method for reducing the dose due to serum potassium are derived from the same abstract prototype method, the system can recognize the analogy between these two findings and draw an analogy to serum potassium when queried about serum calcium.

Third, XPLAIN, like NEOMYCIN, has a mechanism for tailoring its explanations to different types of users. Recall that one of the problems with the explanations produced by the Digitalis Advisor was that they often included many steps that had more to do with programming details than medical reasoning. The XPLAIN system tailors its explanations to different types of users by varying which steps are included in an explanation depending on the type of user. *Viewpoint* markers are attached to steps in a prototype method indicating what types of users the step should be explained to. While generating an explanation for a step, the system checks the viewpoint attachments to see if that step should be included in an explanation to the current user. This approach allows the system to separate those steps that are appropriate for a given audience, such as domain experts, from those that are not.

4.3 Advantages of Explicit Strategic Knowledge

As NEOMYCIN and XPLAIN have demonstrated, explicit representation of strategic knowledge allows improved explanations to be generated. These systems are able to produce abstract descriptions of their problem-solving strategies and XPLAIN is able to justify its behavior. Furthermore, as the example in Figure 9 shows, an abstract representation of problem-solving knowledge allows the system to recognize when the same strategy is applied in different situations. This capability allows XPLAIN to produce explanations by analogy.

In addition, the implementors of these systems have noted that representing control knowledge explicitly and separately from domain knowledge has made the systems more modular and easier to maintain. For example, the XPLAIN system has a domain principle that achieves the goal of checking for digitalis sensitivities. The domain rationale of this domain principle specifies the constraints that a finding must meet in order to be considered a sensitivity. Therefore any new sensitivities that are added to the domain model will automatically be checked by this principle. Moreover, Clancey argues that NEOMYCIN's metarules constitute a domain-independent diagnostic strategy that could be applied to related problems in other domains.[4]

While NEOMYCIN and XPLAIN were significant improvements over their precursors in terms of explanatory capabilities, much of this improvement came from better and more explicit representation of their knowledge and not from increased sophistication in their explanation strategies. These systems still use very simple strategies that do little more than translate metarules or domain principles. Examples like that shown in Figure 8 demonstrate that these simple strategies often produce text that is unsuitable for users. The problem with this explanation is that its structure corresponds too literally to the structure of the method for achieving the task – applying one rule after another. While the explanation repeats that it "gave thorough consideration to any recent findings" *four times*, the explanation does not make clear the overall strategy of applying rules that strongly conclude about the current hypothesis (e.g. causal precursors) before applying rules that are weaker indicators (e.g. general evidence). What is needed is a more sophisticated explanation strategy that could recognize the similarity in the four rule applications and structure this information accordingly.

Furthermore, the explanation capabilities of these systems are still very limited and inflexible. Except for the simple analogies drawn by XPLAIN and a limited capability to tailor explanations to different classes of users, these systems are not capable of producing different explanations to different users or in different contexts. There are still important types of questions that cannot be answered and, except for allowing successive why-questions and the simple "how" mechanism in NEOMYCIN, these systems cannot handle follow-up questions in a meaningful way.

5 More Sophisticated Approaches to Explanation Production

We have seen that richer, more explicit knowledge representation alone is not enough for producing good explanations. Explanation systems that limit themselves to exploiting clever ways of traversing, pruning, and translating the system's execution trace do not produce suitable explanations. We believe that producing good explanations is a complex problem-solving task requiring its own expertise. We advocate the view that explanation must be *decoupled* from the expert system's knowledge representation formalism and problem-solving activity and that explanation requires its own body of knowledge, in addition to the knowledge used by the expert system. In particular, explanation requires knowledge about language, the way language is used to achieve goals, how clauses may be combined to form a coherent text, how responses may be tailored to different user or situations, and rules of conversation.

Although some of the inadequacies of earlier systems stem from their limited natural language *understanding* capabilities, this paper concentrates on work in text planning and generation. The reader is referred to [Leh78] for more information about question

[4] See [Cla83a] and [NSM85] for more details.

understanding and to [CP79], [AP80], [SI81], [Car87], and [LA87] for discussions of how participants in a conversation can infer one another's goals and plans.

This section reviews work in the field of text generation where researchers have developed strategies for producing responses to user's queries based on analyses of naturally occurring text. Section 6 describes our own work in the area of explanation.

5.1 Using Discourse Structures to Generate Explanations

One of our criticisms of the explanation components of previous systems is that the structure of their explanations followed the structure of the reasoning component's solution to a problem, rather than obeying rules of discourse structure. In this section, we consider three efforts that use principles of discourse to generate responses. The principles employed by these systems were derived from empirical observations of naturally occurring explanations and are encoded into strategies for producing texts.

5.1.1 TEXT

In the TEXT system [McK82], McKeown addressed the problem of responding to questions about database entities and structure. From a study of naturally occurring descriptive texts, she devised domain-independent strategies for responding to the following types of questions:

1. Request for definition: *What is a <e>?*
2. Request for available information: *What do you know about <e>?*
3. Request for information about differences between entities: *What is the difference between <e1> and <e2>?*

In developing the TEXT system, McKeown identified two problems: 1) deciding what knowledge is relevant to include in a response, and 2) organizing that information into a coherent text. The main emphases of the TEXT system were to study how discourse structures and focus constraints could be used to guide the information retrieval and generation of multi-sentential responses.

Linguists had previously proposed *rhetorical predicates* that identify the functions played by individual clauses in a discourse [Gri75]. McKeown's analysis of naturally occurring texts showed that certain combinations of rhetorical predicates are more likely to occur than others and that certain predicates are more appropriate in some discourse contexts than others. For example, speakers frequently define or describe objects in terms of their constituent parts in the following way:

1. Identify the object as a member of some generic class or give attributive information about it.
2. Introduce constituents of the object being defined.
3. Provide characteristic information about each constituent in turn.
4. Provide attributive or analogical information about the object being defined.

```
Identification/Attributive
Constituency
Cause-effect*/Attributive*/
    {Depth-identification/Depth-attributive
        {Particular-illustration/Evidence}
        {Comparison; Analogy/Renaming} }+
{Amplification/Explanation/Attributive/Analogy}
```

Figure 10: TEXT Constituency Schema[5]

To encode these standard patterns of discourse structure, McKeown devised several *schemata* that represent combinations of rhetorical predicates. The above pattern is embodied in the *constituency schema* shown in Figure 10. McKeown identified four schemata, each of which could be used to achieve one or more "discourse purposes."

In the TEXT system, discourse purposes correspond to the three question types handled by TEXT. Associated with each discourse purpose is a list of possible schemata that can be used to generate a response that will achieve that purpose. For example, to answer a request for a definition, TEXT can choose one of two schemata: *identification* or *constituency.* Figure 11 shows how TEXT responds to the question "What is a guided projectile?"

When a question is posed to TEXT, the system first selects a subset of the knowledge base that contains all of the knowledge relevant to answering this question, This subset is called the *relevant knowledge pool.* TEXT then chooses a schema based on the discourse purpose and the contents of the relevant knowledge pool. For example, in responding to a request for definition, the relevant knowledge pool contains all of the entity's attributes, superordinates and subordinates. If the relevant knowledge pool contains a rich description of the object's subordinates and less information about the object itself, TEXT uses the constituency schema. Otherwise, the identification schema is chosen.

The schemata themselves contain many options and alternatives. To instantiate a schema, its components are filled-in sequentially by matching the rhetorical predicates against the propositions in the relevant knowledge pool. When more than one instantiation is possible, the system appeals to a set of constraints about how focus of attention can shift to select the most appropriate alternative. Although previous work had defined how focus constraints could aid in understanding texts [Gro77], [Sid79], McKeown was the first to use focus in text generation to control the choice of what can be said next while still obeying rules of coherence. When several alternatives for what to say next exist, the focus constraints dictate selection of the proposition according to the following priorities:

1. Shift focus to a topic just introduced.

2. Maintain the current focus.

[5] The "{}" indicate optionality, "/" indicates alternative, "+" indicates that the item may appear 1 or more times, and "*" indicates that the item is optional and may appear 0 or more times.

(definition GUIDED)

Schema selected: constituency

Predicates instantiated:
(1) identification
(2) constituency
(3) attributive
(4) attributive
(5) evidence
(6) evidence
(7) attributive

(1) A guided projectile is a projectile that is self-propelled. (2) There are two types of guided projectiles in the ONR database: torpedoes and missiles. (3) The missile has a target location in the air or on the earth's surface. (4) The torpedo has an underwater target location. (5) The missile's target location is indicated by the DB attribute DESCRIPTION and the missile's flight capabilities are provided by the DB attribute ALTITUDE. (6) The torpedo's underwater capabilities are provided by the DB attributes under DEPTH (for example, MAXIMUM_OPERATING_DEPTH.) (7) The guided projectile has DB attributes TIME_TO_TARGET_UNITS, HORZ_RANGE_UNITS and NAME.

Figure 11: Sample Response Generated by TEXT

3. Return to a topic previously introduced.
4. Shift focus to a topic with the greatest number of implicit links to the list of topics just introduced.

The TEXT system produces coherent multi-sentential responses to questions about description and comparison of entities. Recall that one of the inadequacies of current expert system explanation facilities is the inability to explain their terminology. The schemas devised by McKeown could be adapted to provide descriptions of the objects in an expert system's domain of expertise. However, the variety of question types that must be handled in an expert system environment is significantly richer. Queries to expert systems frequently require explaining or justifying problem-solving knowledge, and thus the discourse analysis done by McKeown cannot be directly applied to generating such responses.

5.1.2 BLAH

The BLAH system [Wei80] is capable of providing explanations of its reasoning in the domain of U.S. income tax law. BLAH is primarily concerned with structuring explanations so that they do not appear too complex, and thus are easy to understand. In designing

$e \rightarrow \text{AND}\ e\ e\ (e)^n$
$e \rightarrow \text{OR}\ e\ e\ (e)^n$
$e \rightarrow \text{STMT/RSN}\ e\ e$
$e \rightarrow \text{THEN/IF}\ e\ e$
$e \rightarrow \text{EXAMPLES}\ e\ e\ (e)^n$
$e \rightarrow \text{ALT}\ e\ e\ (e)^n$
$e \rightarrow \text{simple text}$

Figure 12: BLAH's Grammar of Explanation

the explanation strategies for BLAH, Weiner studied naturally occurring explanations to discover their structure. Weiner discovered that the important features in creating understandable explanations are: syntactic form, managing the embedding of explanations, and focus of attention. From this analysis, he compiled a "grammar of explanation." The rules of this grammar appear in Figure 12.

BLAH is capable of answering three types of questions:

1. (SHOW x) asks whether the assertion x is believed by the system, e.g., *Must Herby file a tax return?*

2. (CHOICE x y) asks the system to choose between two alternative solutions, e.g., *Should I file the long or short form?*

3. (EXPLAIN x) asks the system to explain why some assertion in the knowledge base is believed, e.g., *Explain why Peter is a dependent of Harry's?*

When the user asks a question, BLAH's reasoning component is invoked. The output of the reasoning component is a tree representing a statement and its support. This tree is produced following the rules of the grammar shown in Figure 12. The non-terminals represent types of justifications and the terminals represent assertions. For example, in the process of reasoning about the query (EXPLAIN (PETER IS A DEPENDENT OF HARRY'S)), BLAH produces the tree shown in Figure 13.

Once the reasoning tree has been produced, the explanation generator translates the tree into text using templates associated with each assertion. However, before the translation is done, the explainer may decide to modify the tree in the following ways to make the final explanation more understandable to the current user:

1. remove all assertions the system can presuppose the user knows from the tree, and

2. determine how much detail should be given in explanation, and if necessary, break up the tree into subtrees.

In order to reason about what the user knows, BLAH segments its knowledge base into the system's (i.e., expert's) view and the user's view. The rules and assertions in the user's view represent a model of the user's knowledge. Any assertion that BLAH can prove using

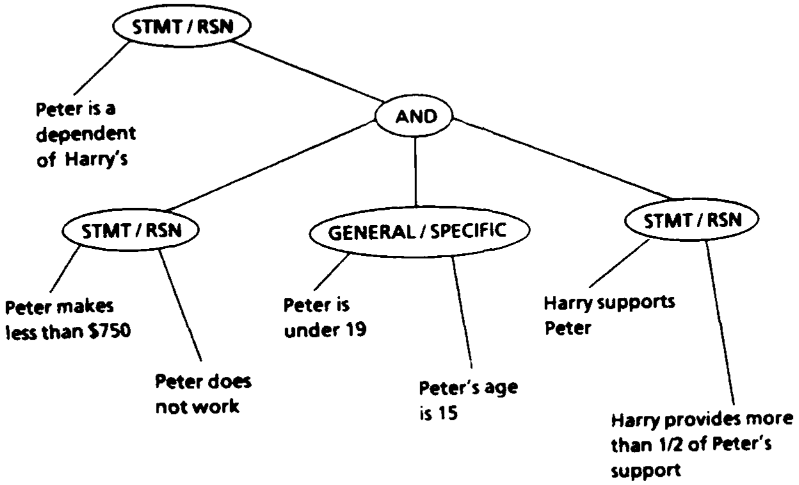


Figure 13: Reasoning Tree of "Peter is a dependent of Harry's"

Peter is a dependent of Harry's because Peter makes less than 750 dollars because Peter does not work and Peter is under 19, in fact Peter is 15, and Harry supports Peter because Harry provides more that half of Peter's support.

Figure 14: Sample Response Generated by BLAH

only information in the user's view (including both assertions and rules) is assumed to be known to the user. BLAH deletes such assertions from the reasoning tree. For more details about this process, see [Wei80].

After the tree has been pruned of knowledge that the user already knows, BLAH must decide how to structure the explanation. To understand why BLAH does not simply traverse the tree from left-to-right translating all of its terminal nodes, consider the explanation that would be generated from the tree in Figure 13 using this process. This explanation, shown in Figure 14, is not very understandable because it is difficult to tell which clauses justify which other clauses. For example, it is difficult to tell whether "Peter is under 19" justifies the statement "Peter is a dependent of Harry's" or the statement "Peter makes less than 750 dollars." The explanation is ambiguous and unless the reader has knowledge of the domain, he cannot disambiguate it. The problem with this explanation is that information has been lost in going from the hierarchical tree representation to the linear representation of natural language text. To make the explanation understandable, this lost information must somehow be put into the English translation. BLAH does this by breaking up complex reasoning trees into subtrees, and adding structural markers to the generated text.

For example, the tree in Figure 13 would be broken up into the subtrees shown in Figure 15. The final explanation generated by BLAH from these subtrees is shown in Figure 16. In this explanation, "uh" acts as a structural marker indicating a shift in focus of

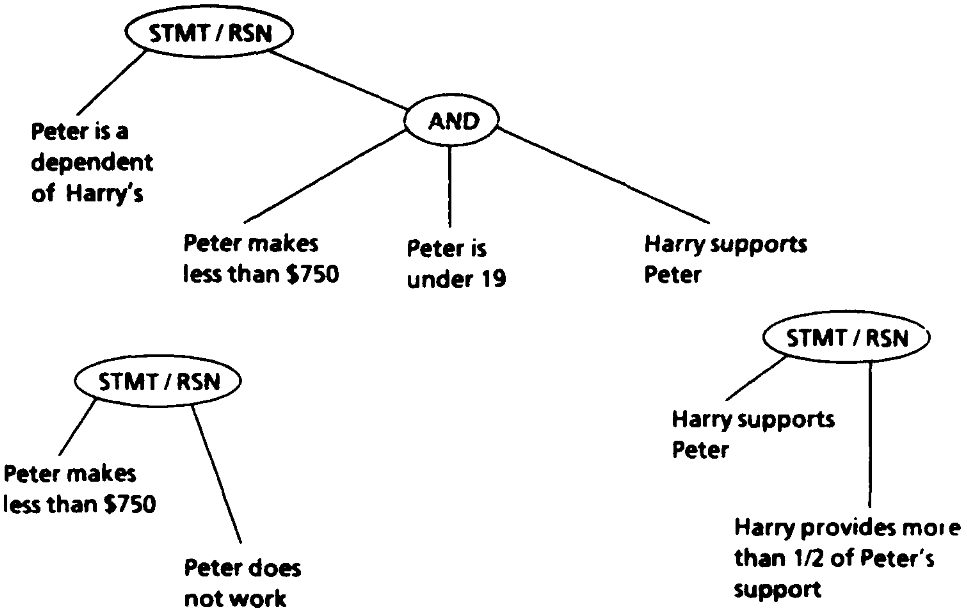

STMT / RSN
Peter is a dependent of Harry's
AND
Peter makes less than $750
Peter is under 19
Harry supports Peter
STMT / RSN
Peter makes less than $750
Peter does not work
STMT / RSN
Harry supports Peter
Harry provides moıe than 1/2 of Peter's support

Figure 15: Simplified Reasoning Trees

Well, Peter makes less than 750 dollars, and Peter is under 19, and Harry supports Peter so Peter is a dependent of Harry's. Uh Peter makes less than 750 dollars because Peter does not work, and Harry supports Peter because Harry provides more than one half of Peter's support.

Figure 16: Sample Response Generated by BLAH

attention back to a previously mentioned item. Repetition of clauses, such as "Peter makes less than 750 dollars," also shifts the focus of attention back to the repeated node. This explanation is easier to understand than the one shown in Figure 14 because the structural markers and repeated clauses give the reader the information necessary to establish the appropriate underlying structure.

5.1.3 ROMPER

From an analysis of transcripts of a radio call-in program in which callers asked the advice of a financial expert,[6] McCoy found that expert-caller interactions frequently involve the expert correcting the caller's misconceptions about objects in the domain or defining terms. In the ROMPER system, McCoy has taxonomized the types of misconceptions about ob-

[6] These dialogues were collected and analyzed by Pollack et al. [PHW82].

USER: I am interested in buying some corporate bonds. Should I get preferred?

SYSTEM: Preferred are not bonds, they are stock. A preferred stock is like a bond in that both have regular payments of a fixed amount. However, unlike bond payments, the company is under no legal obligation to pay preferred stock payments. In addition, preferred stock gives part ownership in the company.

Figure 17: Sample Corrective Response from ROMPER

jects that occur and has devised discourse strategies for producing corrective responses to misconceptions of each type [McC85a].

An example of the type of response ROMPER is capable of producing is shown in Figure 17.[7] The type of misconception that has occurred in this sample dialogue is a *like-super* misconception and it is a kind of *misclassification* because the user has posited that the object (preferred) belongs to an incorrect class (bonds). This type of misconception often occurs when the object and the posited superordinate class share some salient properties, in this case, regular-timed payments and fixed-amount payments.

Based on her analysis of expert-caller interactions, McCoy noticed that corrective responses to this type of misconception usually contain the following information:

1. Deny the incorrect classification.
2. State the correct classification.
3. Concede the similarity of the questioned object to the posited superordinate by mentioning their common attributes.
4. Override the conceded information by offering features that "define" the object as belonging to its correct superordinate class.

McCoy encoded this pattern into the schema shown in Figure 18.

ROMPER's schemas differ from those of McKeown's TEXT in several ways. McCoy's schemas indicate structural relationships between the propositions in a text, but do not necessarily specify an ordering of the propositions as was the case in TEXT. In addition, ROMPER's schemas explicitly encode *two* kinds of information: 1) specification of the content of each proposition, e.g. "classification," "share-attributes" and 2) information about the *communicative role* that each proposition plays in the strategy, e.g. "deny," "state," "concede," "override". McKeown's TEXT schemas encode only the first type of information. Because of the separate and explicit representation of communicative function and content, McCoy's schemas hold more promise for adapting to dialogue capabilities, which we discuss in Section 6.

[7] ROMPER did not take English input or produce English output. We show the translation here for readability.

```
((deny (classification OBJECT POSITED))
 (state (classification OBJECT REAL))
 (concede (share-attributes OBJECT POSITED ATTRIBUTES1))
 (override (share-attributes - POSITED ATTRIBUTES2))
 (override (share-attributes OBJECT REAL ATTRIBUTES3))
```

Figure 18: ROMPER Like-Super Schema

5.2 Tailoring Explanations

Another criticism of existing explanation facilities is that they present the same explanation to all of the people all of the time. They do not adapt their explanations to the user's goals, to his knowledge of certain concepts in the domain, to his level of expertise, to his preferences, or to his perspective. As we move to more sophisticated expert system frameworks, the problem of selecting what to say becomes more critical, because the richer knowledge bases that these frameworks support present a broader range of options.

As we observed in the previous section, both NEOMYCIN and XPLAIN had mechanisms to tailor their explanations to different classes of users, but their techniques are very primitive implementations of stereotypic user models as proposed by [Ric79]; neither is a general solution. Their simple techniques again place the responsibility for functionality in the explanation capability on the programmer, since the knowledge base has to be marked *a priori* in some way to indicate how it should be explained to different classes of users. This marking is then fixed for each class. All classes must be defined when the system is built and all of the knowledge that is to be explained differently to different users appropriately marked. The explainer has no flexibility in deciding when to use different terminology or to go to a different level of detail.

In an attempt to address this limitation, some recent research has begun to address the issues of what dimensions explanations can be varied along; how knowledge representation must be augmented to handle the various dimensions and their interactions; and how an explainer can decide when and along what dimension an explanation should be tailored, and how that interacts with planning text. This section considers systems that have flexible explanation facilities and the capability to vary their explanations for different users or in different situations.

5.2.1 Varying Verbosity

Lehnert's QUALM [Leh78] has a mechanism for varying how much information goes into an answer. The system contains a set of heuristics called *elaboration options* that allow the system to elaborate its responses depending on the *attitudinal mood* of the system, the type of question asked, and the results of an initial memory search. The attitude of the system is represented by a value from a partially ordered set of mood values ranging from talkative to sarcastic. A sample elaboration heuristic is shown in Figure 19.[8] This elaboration heuristic

[8] This is a simplification of the option as it appears in [Leh78].

```
Correction/Explanation Option
IF: 1) mood = cooperative or talkative,
    2) question type = verification, and
    3) initial response = no
THEN: 1) reformulate question into a concept completion
      2) look for answer to the concept completion question
      3) if an answer is found, append it to the "no" response
```

Figure 19: An Elaboration Option

allows a response to be varied according to the mood of the system as in the following:

Q:	Did the waitress give John a menu?
A1:	No. *(mood = minimally responsive)*
A2:	No, the hostess gave John a menu. *(mood = cooperative)*

Note that the use of elaboration heuristics places the responsibility for deciding when and how to elaborate a response on the explanation strategies. The knowledge base is not marked *a priori* to say what knowledge is an elaboration of what other knowledge. These decisions are made at question/answering time, depending on the type of question asked and the results of a preliminary memory search.

Only a few of the elaboration heuristics posed by Lehnert were actually implemented in QUALM, and many of the others were only vaguely specified. Furthermore, the mood of the system is not sufficient to determine when something should be elaborated. For example, answer A1 above will actually confuse the questioner in certain circumstances (e.g., if the user is interested in knowing whether or not John has been given a menu and not so much in who gave it to him.) Therefore, other considerations should be taken into account in making the decision to elaborate.

5.2.2 Tailoring to User's Perspective

In the ADVISOR system, McKeown et al. are concerned with being able to generate an explanation tailored to the user's goal as determined from the previous discourse. Being able to generate tailored explanations requires the system to be capable of generating different explanations of the same information or advice. She has identified four dimensions of explanation which can each be varied in an individual response: point of view, level of detail, discourse strategy, and surface choice [McK84].

McKeown et. al. have concentrated on tailoring explanations to different *points of view* [MWM85]. They have developed techniques for representing various points of view in the system's knowledge base. From an ongoing dialogue, ADVISOR infers the user's goal and then uses this goal in selecting a point of view for tailoring the content of its explanations.

In ADVISOR's domain of student advising, there are a number of different points of view that the student may adopt when selecting courses and explanations should be tailored appropriately. For example, in answering the question

Q: Should I take discrete math and data structures this semester?

the following response is appropriate if the student's goal is to complete the requirements as soon as possible:

A1: Yes, data structures is a requirement for all later Computer Science courses and discrete math is a co-requisite of data structures.

while the following is an appropriate anפwer if the student is ^oncerned about completing requirements at the proper stage in the program:

A2: Yes, you usually take them both first semester, sophomore year.

However, neither of these is an appropriate response if the student's goal is to be able to take courses that match his personal interests. A more appropriate response, assuming the student was interested in Artificial Intelligence, would be:

A3: Yes, if you take data structures this semester, you can take Introduction to AI next semester, and you must take discrete math at the same time as data structures.

To select a perspective to use in an explanation, ADVISOR infers the user's current goal from the discourse. McKeown's work on goal inferencing extends previous work in this area, since her model derives not only the goal of each individual utterance (as in [AP80]) but also infers a higher-level goal relating the goals of a *series* of utterances.

Once the user's current goal has been inferred, the problem then is to find information in the knowledge base that is relevant to the selected perspective. The knowledge base is organized as intersecting multiple hierarchies, which are linked by the entities that can be viewed from different perspectives, e.g.,courses in the student-advising domain. For example, to produce response A1 above, the system would extract information about the relationship between data structures and discrete math from what McKeown calls the *requirements hierarchy*, while to construct response A2, it would get its information from the *state model hierarchy*. This partitioning of the knowledge base allows the explainer to distinguish between different "types" of information that support the same fact.

Using the information from the appropriate hierarchy, a production system derives a response to the user's question. Information from one hierarchy will cause different rules to be applicable than will information from another hierarchy. The reasoning trace that the production system leaves behind is used as a basis for producing the explanation. Since ADVISOR uses different knowledge in its reasoning process depending on the user's goal, the system can produce different responses to the same question, or different justifications for the same response as dictated by the inferred goal. For example, ADVISOR is capable of producing responses A1 through A3. Each of these provides a different justification for the same advice tailored to the user's goal.

McKeown's work concentrates on using different points of view in providing justifications for advice. As part of ROMPER, McCoy [McC85b] has developed a notion of *object perspective* for deciding what information to include when providing corrective responses to misconceptions about objects.

McCoy points out that while two objects may be considered similar when viewed from one perspective, they may be very different when viewed from another. Consider the following sample dialogues from [McC85b]:

CLIENT We have $40,000 in Money Market Certificates. One is coming due next week for $10,000.... I was wondering if you think this a good savings....

EXPERT Well, I'd like to see you hold that $10,000 coming due in a Money Market Fund and then get into a longer term Money Market Certificate.

CLIENT Hmmm.... well I was just wondering, what about a Treasury Bill instead?

EXPERT That's not a bad idea, but it doesn't replace your Money Market Certificate in any way - it's an exact duplicate. They're almost identical types of instruments - so one, as far as I'm concerned, is about the same as another.

Now consider how the same two objects can be seen to be very different when viewed from a different perspective:

CLIENT I am interested in buying some U.S. Government Securities. Now I was thinking of Money Market Certificates since they are the same as Treasury Bills.

EXPERT But they're not - they are two very different things. A Treasury Bill is backed by the U.S. Government. You have to get it from the Federal Reserve. A Money Market Certificate, on the other hand, is backed by the individual bank that issues it. So, one is a Government Security, while the other is not.

In the first exchange, both objects are viewed as savings instruments. From this perspective the attributes that are important for comparison are interest rates and maturity dates. From this perspective, Treasury Bills and Money Market Certificates are "identical." However, in the second exchange, the objects are being viewed in terms of their issuer. The attributes that are important in this case are the issuing organization and the place of purchase. From this point of view, Treasury Bills and Money Market Certificates are very different.

McCoy has devised an alternative technique for representing perspective. In her scheme, perspective is not represented as different superordinates in a generalization hierarchy (as was done in McKeown's system). Instead, McCoy represents perspective in a structure that is orthogonal to the generalization hierarchy. A perspective is a set of attributes and their corresponding salience values. Salience values (low, medium, high) indicate how important each attribute is in that perspective. Perspectives must be defined *a priori* for the objects in a particular domain and are considered to be a part of the expertise in that domain. Using this representation, when an object is viewed from a particular perspective, the perspective essentially acts as a filter on the properties that the object inherits from its superordinates. Superordinates that do not contribute any of the attributes currently in perspective can be ignored.

McCoy's system assumes that one perspective is active at any point in a discourse and that the active perspective could be determined from preceding discourse, although she does not give a technique for doing so. Perspective is used to provide a similarity measure for finding an object that is similar to the incorrect object posited in the user model and thus to classify the type of misconception that occurred.

```
** EXPERTISE 3
** HOW DOES HYPERPARATHYROIDISM CAUSE RENAL STONES?
     Renal stones are caused by hypercalcemia
          Hypercalcemia is caused by hyperparathyroidism

** EXPERTISE 6
** HOW DOES HYPERPARATHYROIDISM CAUSE RENAL STONES?
     Renal stones are caused by increased urinary calcium
          Increased urinary calcium is caused by hypercalcemia
               Hypercalcemia is caused by increased bone breakdown
                    Bone breakdown is caused by increased osteoclast activity
                         Increased osteoclast activity is caused by hyperparathyroidism
```

Figure 20: Explanations Tailored to Level of Expertise

5.2.3 Tailoring to User's Level of Expertise

Another consideration to be taken into account when generating an explanation is the user's level of expertise. Wallis and Shortliffe [WS84] developed a technique for generating explanations of causal chains that are customized to the user's level of expertise and the amount of information he would like to obtain. Both the user's level of expertise and his desired level of detail are represented as an integer ranging from 1 to 10. The user must declare these two values, although the level of detail can be defaulted to a value compatible with the user's level of expertise. The user can change his declared level of expertise at any time during the interaction. He can request more detail after a given explanation by increasing the detail level and posing the question again.

In order to tailor explanations, the explainer must be able to determine what knowledge should be explained to users of various levels of expertise and what knowledge is considered more or less detailed. To encode this information, Wallis and Shortliffe associate a measure of *complexity* with each inference rule and concept in their knowledge base. However, because some concepts are "key" ideas in a reasoning chain and should be mentioned regardless of their complexity rating, a measure of *importance* is ascribed to each concept as well. These measures are also represented as an integer between 1 and 10.

When generating an explanation, the user's expertise level and calculated detail values act as lower and upper bounds on the complexity of concepts that will be included in the explanation. Terminal concepts (those at either end of a reasoning chain) and concepts whose importance rating is very high are included even if their complexity measure falls outside the specified range. Rule complexity is also taken into account. If two concepts are linked by a rule deemed too complex for the current user's level of expertise, canned text associated with the rule is included instead.

The effect of using the user's level of expertise and a measure of detail on the system's explanations is demonstrated in the example explanations shown in Figure 20 from [WS82].

Note that that only variation in these explanations is whether or not a step in the causal chain is included in the explanation or, in some cases, whether a canned simplified justification of a rule will be substituted for one that is deemed too complex. This scheme is based on the assumption that experts would like to see more of the steps in a causal chain than novices. We question whether this is actually the case. After all, it seems that experts are more likely to be able to fill in the details themselves and it is the novices who need all of the steps spelled out for them.

Clearly there are other considerations that need to be taken into account when distinguishing between novices and experts. Paris has begun to explore some of these in her work on the TAILOR system [Par87]. TAILOR is capable of tailoring descriptions of complex physical objects to the user's level of expertise. To determine what kind of information should be included in explanations to users with different domain expertise, Paris analyzed encyclopedia entries about various complex physical objects [Par85]. For each object, she compared the descriptions given in adult and junior encyclopedias. She found that in addition to varying the *amount* of detail given to different types of readers, the naturally-occurring descriptions varied according to the *kind* of information provided. She noted that if a user is assumed to be very knowledgeable about the domain, objects are described in terms of their subparts and properties. However, if a user has little knowledge of the domain, object descriptions focus on how the object works.

To produce tailored descriptions, Paris employs two distinct discourse strategies. For adults (experts), a part-oriented description is produced using the *constituency* schema devised by McKeown for use in her TEXT system (recall Figure 10.) For naive users, however, Paris formalized a discourse strategy that guides the system in tracing causal links in the underlying knowledge base to produce the text.

Paris does not assume that users fall into one of several stereotyped classes, e.g. novice vs. expert. Rather, she views "naive" and "expert" as the extremes of a knowledge continuum, where most users fall somewhere in between since they may have *local expertise* about some objects in the knowledge base while being naive about others. The TAILOR system assumes a user model that contains knowledge of which specific objects in the knowledge base are known to the user as well as an indication of whether or not the user understands the "underlying basic concepts." Given such a model, TAILOR can mix the two discourse strategies to cover cases where the user has significant knowledge about some aspects of the object being described and is naive about others. For example, Figure 21 shows TAILOR's description of a telephone. In this example, taken from [Par87], the user model indicates that the user knows about loudspeakers, but does not know about microphones or how they work in conjunction as a telephone. Because the user knows about one of the two parts of the telephone (a receiver is a kind of loudspeaker), the constituency schema is selected. However, before providing structural information about each of the subparts, the system consults the user model and learns that the user has no local expertise about transmitters (a type of microphone) and so the system switches to the process trace strategy to describe the transmitter.

The telephone is a device that transmits soundwaves. The telephone has a housing that has various shapes and various colors, a transmitter that changes soundwaves into current, a curly-shaped cord, a line, a receiver to change current into soundwaves and a dialing-mechanism. The transmitter is a microphone with a small diaphragm. A person speaking into the microphone causes the soundwaves to hit the diaphragm of the microphone. The soundwaves hitting the diaphragm cause the diaphragm to vibrate. The vibration of the diaphragm causes the current to vary. The current varies, like the intensity varies. The receiver is a loudspeaker with a small aluminum diaphragm. The housing contains the transmitter and it contains the receiver. The housing is connected to the dialing-mechanism by the cord. The line connects the dialing-mechanism to the wall.

Figure 21: Description Generated by Mixing Two Strategies

5.3 Dialogue Capabilities

Recall that another limitation of expert system explanation facilities is that they have limited or no dialogue capabilities. Much of the work done in the area of discourse has focused on *understanding* certain aspects of dialogues (e.g., tracking the topic of a dialogue [Gro77], resolving anaphora [Sid79], handling interruptions [GS86], and recognizing subdialogues [Lit85]). The important difference between interpretation and generation is that in interpreting a dialogue a system does not have to make choices since the utterances that follow indicate what choice was made. As we have already seen, generating a response requires a system to determine what information to include in the response and to choose a strategy for presenting the information. There has been surprisingly little work in the area of providing question/answering systems with dialogue capabilities. Most systems treat each question-answer pair independently. The types of follow-up questions handled are very limited, and systems that keep a dialogue history and make use of it when responding to subsequent queries are rare.

Some of the early natural language interfaces to databases were capable of handling ellipsis and anaphora by saving the previous query and comparing it to the current one (e.g., [WK72], [Hen77]). If the syntactic and semantic structures matched, the missing part(s) of the current query would be filled in from its predecessor. In a similar fashion, Lehnert's QUALM is able to handle dialogue in a very limited way. Again, the last utterance is saved for handling ellipsis. QUALM has some simple conversational continuity rules that refer to this last utterance in producing answers to questions such as:

Q: Did John leave for New York?
A: Yes. (John left for New York.)
Q: When? (When did John leave for New York?)

Q: Is John still in charge?
A: No, Bill is. (Bill is in charge.)
Q: Who's he? (Who's Bill?)

Others have begun to look at how keeping a more extensive discourse history could affect the responses given. Swartout's XPLAIN system remembered the last few questions it had answered and was able to draw simple analogies, like that shown in Figure 9.

As part of her work on the TEXT system, McKeown did some preliminary analysis of how previous discourse might affect the types of responses generated by the system [McK82]. This analysis considered how :nuch and what information should be kept in the dialogue history. She considered two options: (1) the system could remember just that each question had been asked and responded to, or (2) the system could remember that a question had been asked, the structure used in the response, and the particular information provided. McKeown discussed how responses to some questions could be affected by each type of dialogue history. However, no dialogue history was maintained in the implementation, and none of the suggestions for how responses could be altered if such a history were made available were implemented. Furthermore, the analysis was limited to considerations such as avoiding repetitions and generating contrasts or parallels with previous responses. Issues of how to handle follow-up questions, elaborations, clarifications, or misunderstandings were not considered. Section 6 describes our approach to some of these questions.

5.4 Explaining Non-Categorial Knowledge

In section 4, we saw that to produce good explanations, the structure of a knowledge base must reflect explanation concerns. However, structuring the knowledge base properly does not, by itself, guarantee good explanations. Often a good explanation will only present selected portions of a knowledge structure. A good explanation facility needs additional knowledge to know what to present and how to present it.

Throughout this section we have seen approaches to explanation that do not simply paraphrase structures from a system's knowledge base or execution trace. In these systems, producing responses is treated as a complex problem in its own right, *decoupled* from the system's knowledge base and problem-solving activity. These systems bring additional knowledge to bear to produce explanations, e.g. knowledge about how coherent object descriptions are structured, knowledge of what certain types of corrective responses should include, knowledge about what kind of information should be presented to users of different levels of expertise, and so forth.

By both structuring domain knowledge carefully and decoupling explanation from problem-solving, it has been possible to create facilities that explain problem-solving strategies which were previously considered "unexplainable", namely problem-solving strategies using non-symbolic approaches such as Bayesian decision theory or heuristic evaluation functions. In the following section, we describe the explanation facility for the QBKG system which uses an evaluation function to make its decisions. For a discussion of how explanations may be produced from a system using decision theory see [Lan87].

5.4.1 The QBKG System

The QBKG system [BA83] is a backgammon-playing program whose knowledge of backgammon is encoded in a heuristic evaluation function. In order to determine the best move to make, QBKG applies this function to each of the legal moves from the current position and selects the move which maximizes heuristic value. Once QBKG has selected a move, the user is free to ask "Why did you make *that* move instead of *this* move?" In order to answer

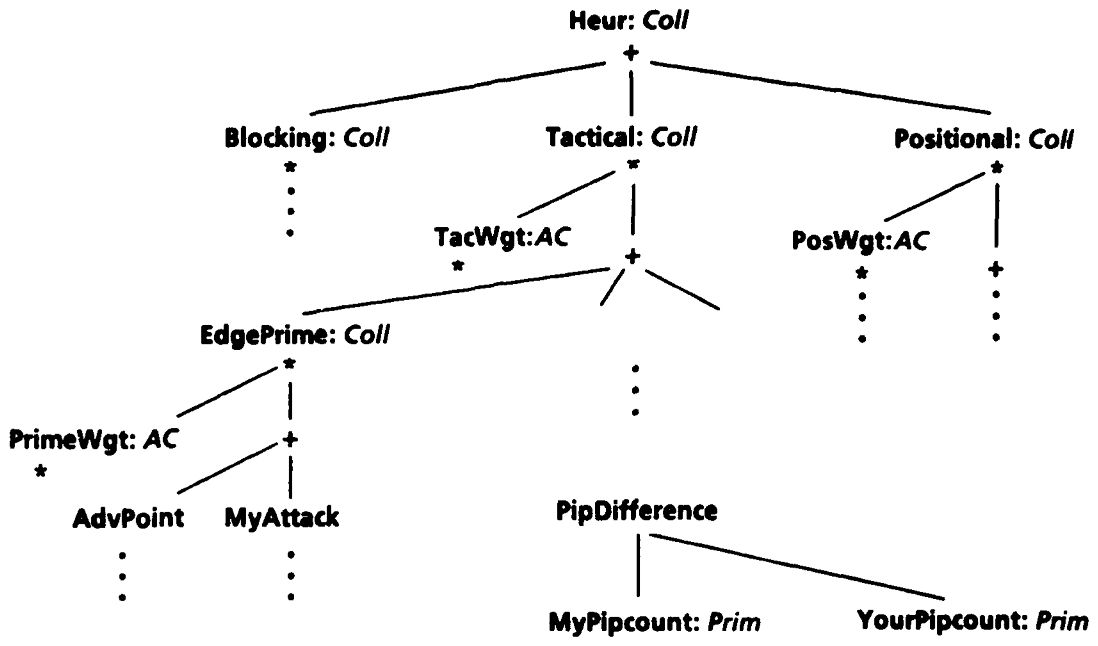


Figure 22: Structure of QBKG's Evaluation Function

this question, QBKG must be able to examine the differences in the values of the evaluation function for the two moves and decide first, when quantitative differences in the evaluation function represent significant differences and what components of the evaluation function contribute to these significant differences, and second, how quantitative differences should be expressed in non-mathematical natural language.

To allow the system to produce such explanations, the explainer must have access to the various components that make up the evaluation function. Moreover, it is crucial that the internal structure of the evaluation function correlates with the structure of backgammon knowledge in a way that is meaningful to people. As shown in Figure 22, QBKG's evaluation function is represented as a hierarchical structure. At the leaves of the tree are *primitive observations* which are directly obtainable from the current board position, e.g. `MyPipcount`, `YourPipcount`. The primitives are combined into meaningful *concepts* using various mathematical operators. For example `PipDifference` is formed by subtracting `YourPipcount` from `MyPipcount`. Related concepts are collected higher in the tree by weighting each concept appropriately and summing the results to produce a new concept. For example, the level just below the top of the tree is made up of the three concepts `Blocking`, `Tactical`, and `Positional`. The top node in the hierarchy represents the heuristic value, `Heur`, of making a given move.

The fundamental assumption of the explanation process is that important differences between two moves will be reflected by "large"[9] differences in the values of the highest level

[9] How QBKG aecides what differences are large and small in a given context is described in [BA82] and

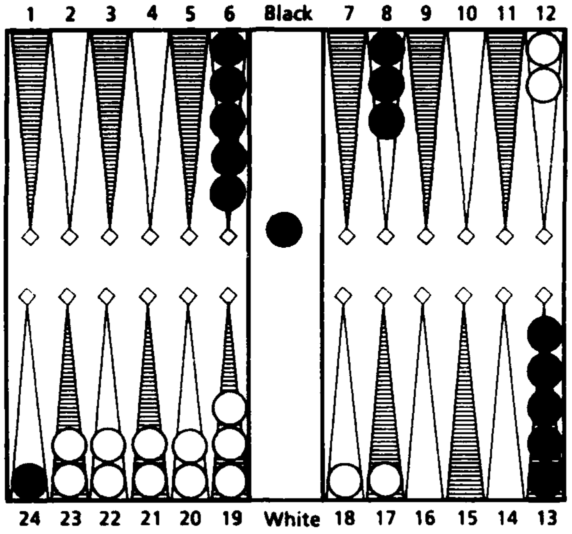
concepts that are related to the differences. To produce a comparison between two moves, the system searches down the tree until it reaches a level where a significant difference is found and *focuses* on the differences in the components at this level. So for example, if the differences for the `Blocking` and `Positional` concepts are small, while the difference in the `Tactical` concept is large, the commentary will focus on comparing tactical aspects of the two moves. Backgammon knowledge related to blocking and positional aspects is assumed not to be relevant and will not be mentioned. In describing the magnitude of the differences, QBKG breaks the space of possible values for each concept up into six classes ranging from "not significantly" to "vastly".

A sample explanation produced by the QBKG system is shown in Figure 23. Since which high level concepts are relevant typically depends on who is ahead in the game and by how much, QBKG's commentaries always begin with a statement of its opinion as to which player has the advantage (determined from a computation of the expected value of the game) and by how much (based on `PipDifference`). This corresponds to Part 1 of the sample explanation. Next, as illustrated in Part 2, QBKG makes a statement about the relative worth of the two moves, based on the difference in the value of `Heur` and some knowledge about the range of possible `Heurs` in this position. Part 3 is the result of the focusing mechanism. The crucial issue of stopping Black from making an advanced point (`AdvPoint`) is discovered and reported while irrelevant differences between the moves, such as the added risk that White may be hit, are ignored. In the sample dialogue, the bracketed numbers in part 3 are reference number by which the user may request further commentary on the specified topics. QBKG responds to such requests by examining the evaluation function with the selected topic as the root of the search tree.

Note that the problem of deciding which differences are significant and should be mentioned in a given context is not limited to systems whose knowledge is encoded in evaluation functions. This problem is faced by systems using symbolic knowledge representations as well. In a symbolic system, while it is trivial to determine when the values of a given attribute of two objects are different, the problem of whether or not that difference is relevant in any given context remains a difficult one. Recall the discussion of in Section 5.2.2 and the difficulty faced by McCoy's ROMPER in determining whether or not two objects should be considered similar from a particular perspective. We return to this problem again in Section 6.2.2.

The results of efforts such as the QBKG system are important because they give us insight into how the behavior of non-symbolic representations may be explained in a non-quantitative way. This is especially important because many problems in domains such as medicine cannot be adequately solved without appeal to inherently quantitative notions such as uncertainty and tradeoffs [LFT*87]. Many problems which resisted solution using symbol-based reasoning systems are now being easily solved with connectionist architectures and results such as those of the QBKG system provide insight as to how we might explain the behavior of these alternative reasoning paradigms.

more extensively in [BA83].



(1) In the given position, White is far ahead in the race, and has
a winning advantage, with substantial gammon chances.
(2) The actual move, 17-23,23-24 (Move 1), is much better than the
suggested move, 12-18, 17-18 (Move 2).
(3) There is nothing to recommend Move 2. The advantages of Move 1 are:
- vastly better chances of keeping Black from making an
advanced point [1].
- very much better attack by White [2].

Figure 23: QBKG's Commentary Comparing Two Moves

6 A Reactive Approach to Explanation

By and large, the approaches described in the previous section view the problem of generating responses as a one-shot process, i.e., they have one chance in which to give the user a correct response that he will understand completely and will not cause him to make any incorrect inferences. At the extreme are those who view text planning as a problem with a derivably "correct" solution, i.e., as a problem to plan a response that will create all and only the correct information in the mind of the hearer. For example, when planning a referring expression, Appelt's KAMP system [App81] attempts to "prove" that as a result of the referring expression, the hearer will be able to successfully identify the correct object in the world. This approach requires a *complete* and *correct* model of their hearer, including not only all of the facts he believes, but also the reasoning strategies he uses in drawing inferences. But is this *realistic*? Sparck Jones has questioned not only the feasibility of acquiring such a model, but also of verifying its correctness, and the tractability of utilizing such a model to affect reasoning and generation of responses [Spa84].

In addition, we question whether such a model is psychologically valid. That is, when people explain things to one another, do they take into account everything their listener knows and all of the inferences he is likely to make as a result of their utterances? If so, one would expect that most interactions proceed from one topic to another, with listeners always understanding what the speaker has said. However, this is inconsistent with the results of analyses of naturally occurring dialogues. In order to determine what type of interface an expert system should ideally provide, Pollack et al. have studied a "naturally-occurring" expert system - a radio talk show in which callers request the advice of a financial expert - and have found that user-expert dialogues are best viewed as a negotiation process [PHW82]. They found that callers rarely stated a problem and listened passively to the expert's advice. On the contrary, they found that the caller and expert often needed to negotiate the statement of the problem to be solved as well as a solution that the caller could understand and accept.

In our own work on explanation, we too have examined samples of naturally-occurring dialogues from several different sources: tape-recordings of office-hour interactions between first year computer science students and their teaching assistants (in which students asked questions about programming concepts and constructs); protocols collected while building the Program Enhancement Advisor [NSM85] of programmers interacting with a mock expert system (in which LISP experts recommended improvements to a user's program and the user was free to ask questions about these recommendations); transcripts of the radio talk show collected and analyzed by Pollack et al. (in which callers ask the advice of a financial expert); and transcripts of electronic mail dialogues collected by William Mann [Rob84] (in which system users present their problems to computer operators and request assistance).

We present two examples from the data, and then state our overall observations.

DIALOGUE 1

[The student and teacher are discussing an assignment to implement an infix calculator using two stacks (arrays) to keep track of the operators and numbers read in. The student is unsure about whether she should store the operator or its precedence on the operator stack.]

STUDENT In the array, for each element, do you store the operator or the precedence?

TEACHER You store the operator.

STUDENT Only?

TEACHER Well, you can find out the precedence, right? From any operator you can find out the precedence. Ok? From the precedence, can you find out the operator?

In this dialogue, the student asks a follow-up question to resolve an incompatibility between her belief (that the precedence must be stored) and the expert's response (that implies it need not be stored.) The expert then elaborates by giving a reason to justify his previous response. Clearly, the expert does not have a complete model of the listener; if he had, he would have anticipated the listener's need for the elaborated explanation and given it the first time around.

DIALOGUE 2

TEACHER OK, so what is it, it's using stacks, right?

STUDENT Ya, well, cause, um, aren't we supposed to use linked lists?

TEACHER You don't have to use linked lists. You don't.

STUDENT But OK. You said stacks, right?

TEACHER In LISP we implemented stacks as linked list. In C we can implement stacks as an array.

STUDENT Wait, in LISP...

TEACHER In LISP, we implemented, past tense, implemented stacks as linked lists. Right? In C, we can do it anyway we want. We can implement it as linked lists or as arrays, uh, I don't know any obvious data structures after that. But, um, you can use a linked list or an array. I would use an array, personally.

In this dialogue, the student does not understand the difference between the general concept of a stack as a way of managing data and its implementation using a particular data structure. The teacher thinks he has cleared up her misunderstanding with his first explanation, but he has not as indicated by the student's "Wait" and hesitation. The teacher elaborates on his earlier response by emphasizing the point he made in the previous explanation, and then elaborating on the notion that a stack can be implemented using various data structures. So, we see that systems must be able to offer further elaborations of the responses or alternate explanations, even when the user is not very explicit about what aspect of the explanation was not clear.

From our analysis the data, we have made several observations:

- *Experts do not have a* **detailed** *model of the user.* From the dialogues we examined, it is clear that the expert does not have a complete and correct model of the user. While we can safely assume that the expert has some model of the user, it seems that since many and varied users are likely to seek his help, this model is more likely to be a stereotypic model that may be incomplete or incorrect for any given hearer than a detailed model of any individual. Yet, as the dialogues show, the user and the expert are able to communicate effectively.

- *Users frequently do not fully understand the expert's response.* Users rarely stated a problem, received a result or explanation, and then left, satisfied that they understood and accepted the expert's explanation. The expert frequently found the need to define terms or establish background information in response to feedback that the listener did not completely understand the response.

- *Users frequently ask follow-up questions.* Users frequently requested clarification, elaboration, or re-explanation of the expert's response. In some cases, follow-up questions take the form of a well-articulated question, either to request justification of a previous response; to ask the expert to compare the expert's solution with an alternative solution the user proposes; to request the definition of a term used in the previous explanation; to resolve inconsistencies between what the user believes about the domain and the information included in the expert's response; or to make certain that the expert has taken all of the necessary constraints into account.

- ***Users often don't know what they don't understand.*** Users frequently could not articulate a clear follow-up question. In many cases, the follow-up was vaguely-articulated in the form of mumbling, hesitation, repeating the last few words of the expert's response, or simply stating "I don't understand." Often the expert does not have much to go on, but must still provide further information.

Thus, we see that there is a real disparity between what the data reveals, on the one hand, and the explanation facilities that current systems provide and the assumptions they make about how users interact with experts, on the other. By taking a one-shot view of explanation production, current systems have overemphasized the role of user modelling, while ignoring the rich source of guidance that people use in producing explanations, namely feedback from thc listener. In our own work, we have abandoned the one-shot assumption in order to make use of that guidance.

We believc that a *reactive* approach to explanation is required – one in which feedback from the user is an integral part of the explanation process. A reactive explanation facility should include the ability to:

- monitor the effects of its utterances on the hearer,
- recover if feedback indicates the listener is not satisfied with the response,
- answer follow-up questions, taking into account previous explanations – *not* as independent questions,
- offer further explanations even if the user does not ask a well-formulated follow-up question, and
- make use of information available in the user model if it exists, but not require it.

6.1 A Reactive Explainer

We have built an explanation component for an expert system which addresses these issues. To provide the capabilities described above, we:

- plan responses such that the intentional structure of the responses is explicit and can be reasoned about,
- use that intentional structure, together with the user's feedback to provide the conversational context needed to plan follow-up explanations,
- taxonomize the types of (follow-up) questions that are asked and understand their relationship to the current context, and
- provide flexible explanation strategies with many and varied plans for achieving a given discourse goal.

Our explanation generation facility is part of the Explainable Expert Systems (EES) framework [NSM85]. When an expert system is built in EES, an extensive development history is created that records the goal structure and design decisions behind the expert system. This structure is available for use by the explanation facility.

We have used EES to construct a prototype expert system, called thc Program Enhancement Advisor (PEA) [NSM85], which we are using as a testbed for our work on explanation

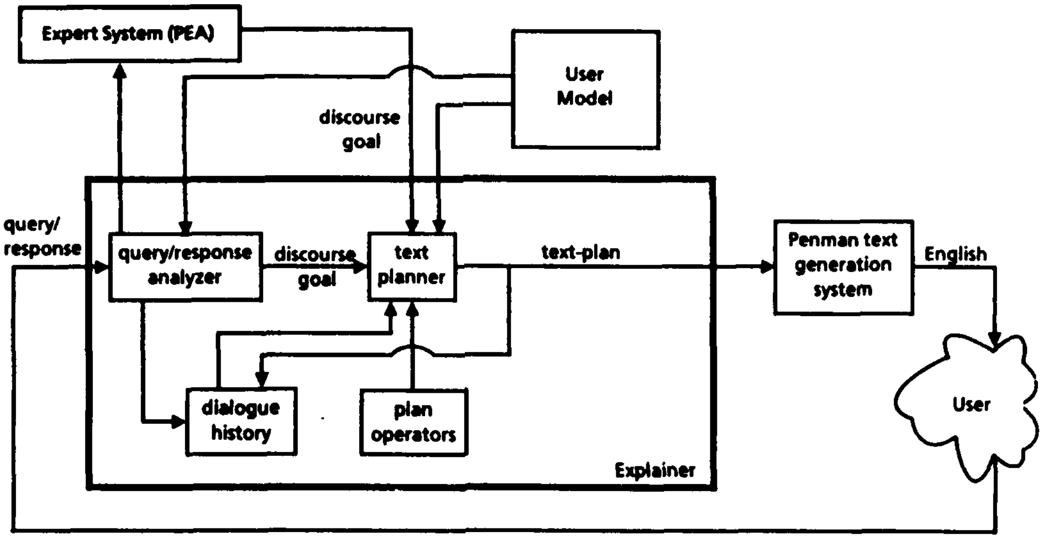


Figure 24: Architecture of Explanation System

generation. PEA is an advice-giving system intended to aid users in improving their Common Lisp programs by recommending transformations that enhance the user's code[10]. The user supplies PEA with the program to be enhanced. PEA begins the dialogue with the user by asking what characteristics of the program he would like to improve. The user may choose to enhance any combination of readability, maintainability, and efficiency. PEA then recommends transformations that would enhance the program along the chosen dimensions. After each recommendation is made, the user is free to ask questions about the recommendation.

An overview of the explanation generation facility (and its relation to the PEA expert system) is shown in Figure 24. To interact with the user, the expert system posts a discourse goal (e.g. persuade the hearer to do an act, describe an entity) to the *text planner*. A discourse goal may be posted as a result of reasoning in the expert system or as a result of a query from the user (see Figure 24). User queries must first be interpreted by the *query analyzer*. Even though we assume the user poses queries in a stylized notation,[11] ambiguities may still arise. An example of an ambiguous follow-up question and the process we use to disambiguate it appears in Section 6.2.1.

Using a top-down hierarchical expansion planning mechanism [Sac75], the text planner plans utterances to achieve discourse goals. When a discourse goal is posted, the text planner searches its library of explanation strategies looking for strategies that can achieve it. A strategy is selected and may in turn post subgoals for the planner to refine. Planning

[10] PEA recommends transformations that improve the "style" of the user's code. It does not attempt to understand the content of the user's program.

[11] To avoid the myriad problems of parsing English-input, we require that the user's questions be posed in a stylized language.

continues in this fashion until the entire plan is refined into primitive operators, i.e. speech acts such as **INFORM, RECOMMEND**.

As the system plans explanations, it keeps track of any assumptions it makes about what the user knows as well as alternative strategies that could have been used to achieve the discourse goals. The result is a *text plan* for achieving the original discourse goal. This text plan is recorded in the *dialogue history* and passed to the Penman text generation system [MM83] for translation into English.

After a response has been generated, the system awaits feedback from the user. This feedback may be a follow-up question (e.g. "Why?", "What is a generalized-variable?"), an indication that the user does not understand the system's response ("Huh?"), or an indication that the user understands and has no follow-up question ("Go Ahead"). The query analyzer interprets this feedback and either returns control to the expert system, or formulates the appropriate discourse goal and passes it to the text planner to produce a response.

If the user asks a follow-up question or indicates that he does not understand the explanation, the system examines the dialogue history to determine how to respond. The information contained there concerning the goal structure of the explanation, assumptions made during its generation, and alternative strategies, is necessary in disambiguating follow-up questions, selecting perspective when describing or comparing objects, and providing furthe.· explanations even when the user does not ask a well-formulated follow-up question.

6.2 Examples

Next, we present several example dialogues and discuss how our approach handles them. These examples were chosen to illustrate how our approach solves some of the problems described in earlier sections.

Consider the sample dialogue with our system shown in Figure 25. While enhancing maintainability, the system recommends that the user perform an act, namely replace `setq` with `setf`.[12] The user, not immediately convinced that this replacement should be done, asks "why?". The query analyzer interprets this question and posts the goal: `(PERSUADE S H (GOAL H Eventually(DONE H replace-1)))` where `S` is the speaker, `H` is the hearer, and `replace-1` is the act of replacing `setq` with `setf`. This is a goal to persuade the user to perform `replace-1`. Discourse goals are represented in terms of the effects that the speaker intends his utterance to have on the hearer.[13]

When a discourse goal is posted, the text planner searches for operators capable of satisfying it goal, i.e. all operators whose *Effect* matches the goal. Each plan operator also contains a *constraint* list, which limits the applicability of the operator; a *Nucleus*, which is a discourse goal for the main topic to be expressed; and optionally, a list of *Satellites* which are discourse goals that express additional information needed to achieve the Effect of the operator.

[12] In many instances PEA is capable of performing the transformation. In such cases, while the actual replacement is done by the system, the user's approval is required.

[13] Following Hovy [Hov88], we use the terminology for expressing beliefs developed by Cohen and Levesque in their theory of rational interaction [CL85]. Space limitations prohibit an exposition of their terminology in this paper. We provide English paraphrases where necessary for clarity.

SYSTEM What characteristics of the program would you like to enhance?

USER Maintainability.

⋮

SYSTEM You should replace `(setq x 1)` with `(setf x 1)`.

USER Why?

SYSTEM I'm trying to enhance the maintainability of the program by applying transformations that enhance maintainability. **Setf**-to-**setq** is a transformation that enhances maintainability.

USER Why?

SYSTEM Transformations that enhance maintainability are defined as transformations whose right-hand-side use is more general than its left-hand-side use. **Setf** can be used to assign a value to any generalized-variable. **Setq** can only be used to assign a value to a simple-variable.

Figure 25: Sample Dialogue 1

```
EFFECT: (PERSUADE S H (GOAL H Eventually(DONE H ?act)))
CONSTRAINTS: (AND (GOAL S ?goal)
                  (STEP ?act ?goal)
                  (BMB S H (GOAL H ?goal)))
NUCLEUS: (FORALL ?goal (MOTIVATION ?act ?goal))
```

Figure 26: Plan Operator for Persuading User to Do An Act

One of the plan operators that matches the current goal is shown in Figure 26.[14] Informally, this plan operator states that if an act is a step in achieving some domain goal(s) that the hearer shares, then one way to persuade the hearer to do the act is to motivate the act in terms of those goals. More formally, this operator's constraints require that there be a *?goal* such that: *?goal* is a goal of the system, `replace-1` is a step in achieving *?goal*, and the speaker and hearer mutually believe that *?goal* is a goal of the hearer. In order to bind *?goal*, the text planner examines the expert system's goal structure. The system assumes that the user shares its top-level goal, `enhance-program`, since he is using the system to perform that task. Furthermore, since the system asks what characteristics the user would like to enhance, the system can assume that the user shares the goal of enhancing those characteristics; in this case, `enhance-maintainability`. The information that the

[14] `(BMB S H x)` should be read as "S believes that S and H mutually believe x." Our plan language makes use of Rhetorical Structure Theory [MT88], a descriptive theory characterizing text structure in terms of the relations (e.g. `MEANS`, `MOTIVATION`) that hold between parts of a text. A detailed description of the plan language is beyond the scope of this paper, see [MP88].

```
EFFECT: (MOTIVATION ?act ?goal)
CONSTRAINTS: (AND (GOAL S ?goal)
                  (STEP ?act ?goal))
NUCLEUS: (INFORM S H ?goal)
SATELLITES: (((MEANS ?goal ?act)))
```

Figure 27: A Plan Operator for Motivating an Act

user shares these two goals is included in the *user model*. In order to avoid explaining parts of the reasoning chain that the user is familiar with, the most specific goal is chosen. Once the constraints have been satisfied, the only possible binding for the variable *?goal* is **enhance-maintainability**.

Once a plan operator has been selected, the planner instantiates it by posting its Nucleus and required Satellites as subgoals to be refined. In this case, since there is only one binding for *?goal*, the single subgoal (MOTIVATION replace-1 enhance-maintainability) is posted. One strategy for satisfying this goal, shown in Figure 27, is to inform the hearer of the goal that the system is trying to achieve (the Nucleus) and then to establish that the act in question is part of the means for achieving the goal (the Satellite). These subgoals are eventually refined to primitive speech acts. The final text plan, shown in Figure 28, is added to the dialogue history and passed to the generator which produces the response shown in the example dialogue.

6.2.1 Disambiguating A Follow-up "Why" Question

After this response is presented, the user asks "why?" a second time. At this point, there are several possible interpretations of this question, including:

I1 Why are we trying to enhance the maintainability of the program?

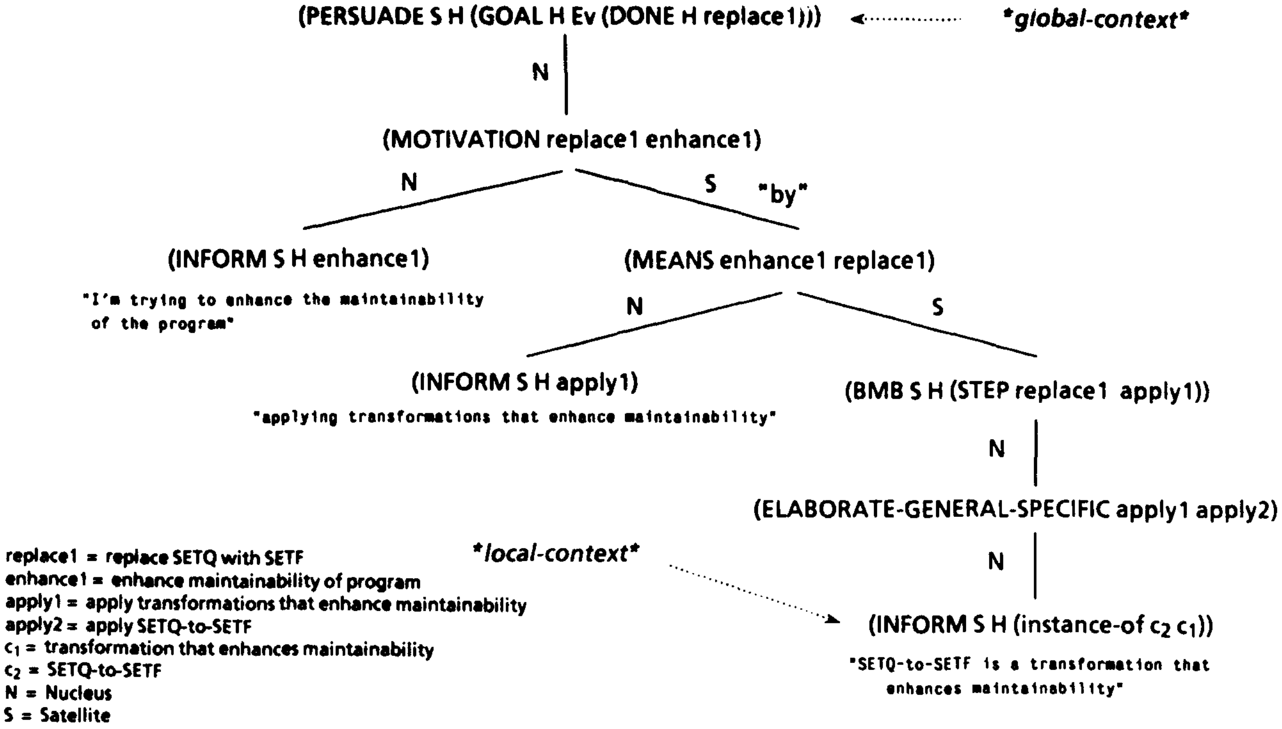
I2 Why are we trying to enhance the maintainability of the program by applying transformations that enhance maintainability? (as opposed to enhancing the program via some other method)

I3 Why is **setq-to-setf** a transformation that enhances maintainability?

This example was constructed to parallel the problematic MYCIN example of Figure 3. Recall that in cases such as this, MYCIN always assumes that "why" is asking why the system is trying to achieve the higher-level goal, corresponding to interpretation I1. This interpretation is often inappropriate. Users are frequently asking for justification of the underlying knowledge, corresponding to I3. MYCIN cannot decide among alternative interpretations because it does not maintain a dialogue history and does not understand the responses it generates.

Resolving ambiguity requires: 1) identifying candidate interpretations, and 2) choosing among them. When multiple interpretations are suggested, our system chooses among them using relevance to the current focus of attention in the dialogue and knowledge about what the hearer knows. This knowledge is embodied in the following two heuristics:

H1: follow immediate focus rules (continuing on the same topic is preferred over returning to a previously mentioned topic, [Sid79], [McK82].)



(PERSUADE S H (GOAL H Ev (DONE H replace1)))
global-context
N
(MOTIVATION replace1 enhance1)
N
S
"by"
(INFORM S H enhance1)
"I'm trying to enhance the maintainability of the program"
(MEANS enhance1 replace1)
N
S
(INFORM S H apply1)
"applying transformations that enhance maintainability"
(BMB S H (STEP replace1 apply1))
N
(ELABORATE-GENERAL-SPECIFIC apply1 apply2)
N
local-context
(INFORM S H (instance-of c2 c1))
"SETQ-to-SETF is a transformation that enhances maintainability"
replace1 = replace SETQ with SETF
enhance1 = enhance maintainability of program
apply1 = apply transformations that enhance maintainability
apply2 = apply SETQ-to-SETF
c1 = transformation that enhances maintainability
c2 = SETQ-to-SETF
N = Nucleus
S = Satellite

Figure 28: Completed Text Plan for Persuading the User to Replace SETQ with SETF

H2: Don't tell the user things he already knows.

Although the tasks of identifying possible interpretations and selecting among them are distinct, it is computationally inefficient to first identify *all* interpretations and then choose among them, as there may be many possible interpretations. Our implementation creates likely interpretations first, and then uses heuristics to rule them out. If an interpretation cannot be ruled out, it is chosen as the interpretation of the user's question and a response is generated. Of course, if our interpretation is incorrect, the user can still recover by asking a follow-up question.

We have found that focus of attention (H1) is a powerful heuristic for ordering the generation of likely interpretations. In the current example, the most recent focus of attention (indicated by the **local-context**) is the statement `Setq-to-setf is a transformation that enhances maintainability`. The system thus infers that the question concerns the rationale behind this statement, unless H2 rules that interpretation out. For example, that interpretation could be ruled out by H2 if the system inferred (based on the user model) that the user already knew why `setq-to-setf` enhances maintainability.

If the first interpretation is ruled out, the system uses the next most recent focus of attention[15] to form the next possible interpretation In this example, that focus refers to the

[15] The text plan records the order in which topics appear in the explanation. This information is used to derive foci of attention in order.

SYSTEM	What characteristics of the program would you like to enhance?
USER	Maintainability.
	⋮
SYSTEM	You should replace **(setq x 1)** with **(setf x 1)**.
USER	What's the difference between **setf** and **setq**?
SYSTEM	**Setf** can be used to assign a value to any generalized-variable. **Setq** can only be used to assign a value to a simple-variable.
USER	What is a generalized-variable?
SYSTEM	Simple-variables are storage locations named by symbols. This concept can be generalized to a storage location that can be named by any accessor function. These storage locations are called generalized-variables.
USER	Huh?
SYSTEM	For example, the *car* and *cdr* of a cons are generalized-variables, named by the accessor functions **car** and **cdr**. Other examples are an element of an array or a component of a structure.

Figure 29: Sample Dialogue 2

means by which the enhancement is being achieved (i.e. **by applying transformations that enhance maintainability**). This leads to interpretation I2. If that interpretation were also ruled out, the system would continue on in this fashion until it found an acceptable interpretation or reached the global context, which, in this case, is the top node in the text plan.

It is important to note that the local and global context, and therefore focus of attention, are derived from the *system's* statement, *not* the user's. The user simply types "why?" in his first two queries in the dialogue; context comes from the response generated by the system. Until now, much work has concentrated on building discourse models that keep track of the user's goals and plans – both domain goals [Car83], [MWM85] and discourse goals [Lit85]. Little work has been done on keeping track of the system's discourse goals and the plans it uses to achieve them (although current work by Grosz and Sidner addresses this issue for the purposes of analyzing dialogues [GS86].) By recording the system's discourse goal and the plan that was used to produce a response, as well as the user's query, our system is able to track conversational context.

6.2.2 Using Context to Select Perspective

Consider the dialogue shown in Figure 29 in which the user asks "What's the difference between **setf** and **setq**?" There are many possible answers to this question if it is considered in isolation:

1. **Setf** has the function name "setf". **Setq** has the function name "setq".

2. None, both can be used to assign a value to a variable.
3. **Setf** is a macro. **Setq** is a special form.
4. **Setf**'s first argument must be a generalized variable. **Setq**'s first argument must be a simple variable.
5. **Setf** can be used to assign a value to any generalized variable. **Setq** can only be used to assign a value to a simple variahle.

Any sophisticated knowledge base representing these concepts will contain much information about **setf** and **setq**. In some ways these two constructs are alike, in some ways they are very different. Deciding which attributes are relevant in which context is a difficult problem. In her work on correcting object-related misconceptions, McCoy [McC85a] recognized this problem and developed the notion of *object perspective*, as described in Section 5.2.2. McCoy assumes that the various perspectives an object can be viewed from are part of the domain knowledge and must be defined *a priori* as part of the knowledge engineering task. In planning a corrective response, she assumes that the perspective to be used in describing the objects the user is confused about is given. She does not provide a means for deciding which attributes belong to a given perspective or which perspective is active at any point in the conversation.

McKeown et al. [MWM85] suggest that the user's goals should be one factor that influences the choice of perspective. Each perspective that an object can be viewed from is indexed by user goals. The system infers the user's potential goal from a series of the user's utterances and uses this goal to determine the perspective that should be used to tailor a response. However, in the last example dialogue, the proper perspective to be used in answering the question cannot be inferred from the user's utterances. Instead, the perspective comes from the system's recommendation, the shared goal (**enhance-maintainability**) and the system's plan for achieving that goal (**apply transformations that replace specific constructs by constructs with more general usage.**)

One of the problems with previous systems that make use of the notion of perspective is that the perspectives must be defined *a priori*. In our system, we do not wish to pre-define which attributes belong to which perspective, but instead to determine which attributes are relevant from the current context: the topics under discussion, the current discourse and domain plans, and the goals that these plans serve. Whether or not an attribute is relevant in the current context is determined by what role it plays in the plan.

In the current example, the system determines the relevant differences in the following way. From the dialogue history, the explainer determines that it has just recommended replacing **setq** with **setf**. It then examines the expert system's problem-solving knowledge, to find the shared goal that caused this recommendation to be made, namely **enhance maintainability** and determines what roles **setq** and **setf** play in achieving the goal **enhance-maintainability**. In this case, the system is enhancing maintainability by applying transformations which replace a specific construct with one that has more general usage. **Setq** has a more specific usage than **setf** and thus the comparison between **setq** and **setf** should be based on the generality of their usage. The system thus generates the response shown in Figure 29.

```
EFFECT: (BMB S H (KNOW H ?concept))
CONSTRAINTS: (AND (SUBCLASS ?sub-concept ?concept)
                  (BMB S H (KNOW H ?sub-concept))
                  (IMMEDIATE-SUBCLASS ?concept ?super-concept))
NUCLEUS: ((SETQ ?diffs (ESSENTIAL-DIFFERENCES ?sub-concept ?concept))
                  (BMB S H (DETAILS-OF ?subconcept ?super-concept ?diffs)))
SATELLITES: (((ABSTRACTION  ?sub-concept ?concept ?super-concept ?diffs)))
```

Figure 30: Plan Operator for Describing an Object by Abstraction

6.2.3 Answering a Vaguely Articulated Follow-Up Question[16]

In Figure 29, the user then asks the question,"What is a generalized-variable?". The query analyzer interprets the question and posts the discourse goal `(BMB S H (KNOW H generalized-variable))`, i.e. the speaker wishes to achieve the state where the speaker and hearer mutually believe that the hearer knows the concept `generalized-variable`.

The system has several plan operators for achieving such a goal. It may describe a concept by describing its attributes and its parts, by drawing an analogy with a similar concept, by giving examples of the concept, or by generalizing a concept the user is familiar with. The plan operator for the latter is shown in Figure 30.

To choose from among these candidate plan operators, the planner has several selection heuristics, including:

SH1 Prefer operators that require making no assumptions about the hearer's beliefs.

SH2 Prefer operators that make use of a concept the hearer knows.

SH3 Prefer operators that make use of a concept mentioned in the dialogue history.

In this case, the user model indicates that the hearer knows the concept `simple-variable`. Hence the operator in Figure 30 requires making no assumptions about the hearer's knowledge, makes use of a concept the user knows, and uses a concept previously mentioned in the dialogue. Thus it is ranked highest by the plan selection heuristics. The final text plan for this example first describes `simple-variables` and then abstracts this concept to introduce `generalized-variables`. This produces the response shown in the sample dialogue.

The user then indicates that he does not understand this explanation with the vaguely-articulated follow-up, "Huh?". From our analysis of naturally occurring dialogues, we devised a set of *recovery heuristics* for responding when a user indicates misunderstanding, but does not ask a well-formulated question. These include:

RH1 If the discourse goal is to describe a concept, give example(s).

RH2 If the discourse goal is to describe a concept, and there is an analogous entity that the hearer knows, draw an analogy to the familiar concept.

RH3 If another plan exists for achieving the discourse goal, try it.

RH4 Expand any unexpanded optional satellites in previous plan operators.[17]

[16] The example described in this section is discussed more fully in [Moo89].

[17] Plan operators may contain optional satellites. Depending on other considerations during plan expansion, some of these may not be expanded. See [MS88].

RH1 and RH2 apply in the context of a particular discourse goal, namely describing a concept, while the other heuristics are more general. The system tries to apply its most specific knowledge first. In this case, RH1 applies and the explainer recovers by giving examples.

This example brings up another interesting point, since it begins with the the user asking about a term used by the system. All explanation systems face the problem of phrasing their explanations in terms their users will understand. Without a complete and correct user model, the only way for a system to guarantee that the terms used in its explanations will be understood is for the system to laboriously ask the user if he is familiar with each term before it is used. This is the only recourse available to one-shot systems. However, because our system is able to elaborate or clarify previous explanations, our system can make assumptions about whether or not the user knows a certain term and recover later if feedback indicates the explanation was not understood.

As illustrated in Figure 30, constraints on plan operators often refer to the state of the hearer's knowledge. The user model includes the domain concepts and problem-solving knowledge, i.e. goals and plans, assumed to be known to the current user. However, the system does not require that this model is be either complete or correct. Therefore, the user model may contain concepts the user does not actually know or omit concepts the user does know. To satisfy a constraint on an operator, the system may assume that a concept is known to the user even if it is not indicated in the user model. As described above, when such an assumption is made, the selection heuristics give the operator a lower rating. If the operator is selected, the fact that an assumption was made is recorded in the plan structure. The system must keep track of such assumptions because these are likely candidates if a misunderstanding occurs later. This leads to another recovery heuristic:

RH5 If any assumptions were made in planning the last explanation, plan responses to make these assumptions true.

For example, in producing the above response about the differences between **setf** and **setq**, the system made the assumption that the user knew what generalized-variables were and simply used this term in the explanation without describing it further. If the explanation were misunderstood, the system would have planned a response to make this assumption true by describing generalized variables.

6.3 Current Status

The expert system and explanation facility described in this section are implemented. There are approximately 50 plan operators, 5 plan selection heuristics, and 5 recovery heuristics. The system can produce the text plans necessary to participate in the dialogue shown and several others that are similar. We are currently in the process of interfacing the text planner to the text generation system.

7 Summary

We have argued that expert systems explanation facilities are inadequate in many ways. We saw that early approaches to explanation were inadequate in part because much of the knowledge needed to answer users' questions was not explicitly represented. For example, early systems did not include justifications of their rules or an explicit representation of their problem-solving strategies. Thus systems such as MYCIN and the Digitalis Advisor could explain *what* they were doing, but not *why* they were doing it.

NEOMYCIN and XPLAIN demonstrated that providing the missing knowledge in an explicit way improved the explanations that could be produced. These systems were able to generate abstract descriptions of their problem-solving strategies and were able to justify their behavior. However, these efforts also showed that augmenting the knowledge bases of expert systems is not, in itself, sufficient to overcome the limitations of simple generation techniques. Explanations produced by these systems were not structured according to rules of discourse, the systems had only limited capabilities for tailoring explanations to different users, and follow-up questions could not be handled in a meaningful way.

Thus we saw that increased sophistication in the system's explanation strategies was needed as well. Simple techniques such as canned text or translating the system's problem-solving activity simply were not satisfactory. To find more sophisticated approaches, we turned to some of the recent work in producing responses to users' questions from the natural language generation literature. These efforts were capable of producing coherent multi-sentential texts that followed rules of discourse and had the capability to tailor responses to the user's goals or level of expertise. However, none of them provide what analyses of naturally occurring text reveal to be an important component of any expert system explanation facility – the ability to take part in an interactive dialogue with the user.

Current expert systems treat explanation in a one-shot fashion. Given a query, these systems respond with the only explanation they can produce. This unnatural approach to explanation depends critically on the quality of the user model and is seriously degraded if that model is incomplete or incorrect. These systems do not keep a dialogue history, but even if they did, they do not have alternative strategies for producing responses or heuristics for deciding when different strategies are appropriate. Because they fail to support dialogue, these systems cannot clarify misunderstood explanations, elaborate on previous explanations, or respond to follow-up questions in the context of the on-going dialogue.

As an alternative, we developed a reactive model of explanation – one that allows for feedback from the user about the understandability of the explanations it produces. By recording the intentional structure of the responses it produces, our system can reason about its own responses when feedback from the listener indicates that an explanation was not understood. This allows our system to respond to follow-up questions in context and react to the user's need for elaboration. We believe that such an approach more closely matches the behavior observed in naturally occurring dialogues.

We believe that explanation for expert systems offers a challenging arena for researchers in many disciplines. The needs of explanation push the boundaries of expressive capability and computational complexity in knowledge representation. Current systems are only able to answer a fraction of the many types of questions users would like to ask, and empowering systems with the ability to answers new types of questions will require enhancements to their knowledge bases. Explanation also provides an interesting domain for work in reactive planning and execution monitoring since ideally a system should be able to produce explanations incrementally, monitoring the user's feedback and allowing the user to interrupt if he is not following or the system is giving him information he already knows. The system should be able to dynamically alter its explanations as a result of this feedback. In addition, explanation provides a rich domain for research in computational linguistics and human-computer interaction. The knowledge bases of expert systems are large and complex and the same knowledge may be represented at various levels of abstraction or in various forms. In answering the many types of questions that may be asked about such a knowledge base, an expert system explanation facility must choose what to include from all the available knowledge, select the appropriate level of detail for a given situation, organize the knowledge into a coherent form, and phrase the knowledge in a manner that will not over- or under-inform a particular user. In some domains, explanations may best be presented by a combination of various media. For example, to explain how a mechanical device works, it may be best to present a pictorial display of the device along with text that describes how it works. Integrating text with graphic displays and narrating animations are challenging open problems. Explanation may also provide a fruitful domain for machine learning since we would like a system to learn new strategies as a result of its explanation experiences and feedback from the user. For example, a system could learn which strategies worked well for users of certain types, which aspects of its domain knowledge are particularly difficult to understand, or what are the common misunderstandings for certain aspects of its domain knowledge and how these misunderstandings are typically corrected. We believe that tackling issues such as these will further improve the explanatory capabilities of expert systems and will produce results that are of general interest in many other areas of artifical intelligence as well.

Acknowledgements

The research described in this paper was supported by the Defense Advanced Research Projects Agency (DARPA) under a NASA Ames cooperative agreement number NCC 2-520. The authors would like to thank Cécile Paris, Margot Flowers, Eduard Hovy, Jasmina Pavlin, Stephen Smoliar, Jay Myers, and Lewis Johnson for their comments and suggestions on an earlier version of this paper.

References

[AP80] James F. Allen and C. Raymond Perrault. Analyzing intention in utterances. *Artificial Intelligence*, 15:143–178, 1980.

[App81] Douglas E. Appelt. *Planning Natural Language Utterances to Satisfy Multiple Goals.* PhD thesis, Stanford University, 1981.

[BA82] Hans J. Berliner and David H. Ackley. The qbkg system: generating explanations from a non-discrete knowledge representation. In *Proceedings of the Second National Conference on Artificial Intelligence*, Pittsburgh, Pennsylvania, August 18-20 1982.

[BA83] Hans J. Berliner and David H. Ackley. *The QBKG System: Knowledge Representation for Producing and Explaining Judgements.* Technical Report CMU-CS-83-116, Carnegie-Mellon University Department of Computer Science, 1983.

[BS84] Bruce G. Buchanan and Edward H. Shortliffe. *Rule-Based Expert Systems: The MYCIN Experiments of the Stanford Heuristic Programming Project.* Addison-Wesley Publishing Company, 1984.

[Car83] Sandra Carberry. Tracking user goals in an information-seeking environment. In *Proceedings of the Third National Conference on Artificial Intelligence*, pages 59–63, Washingtion, D.C., August 22-26 1983.

[Car87] Sandra M. Carberry. Pragmatic modeling: toward a robust natural language interface. *Computational Intelligence*, 3(3):117–136, August 1987.

[CL81] William J. Clancey and Reed Letsinger. Neomycin: reconfiguring a rule-based expert system for application to teaching. In *Proceedings of the Seventh International Joint Conference on Artificial Intelligence*, pages 829–836, IJCAI, Vancouver, B. C., Canada, 1981.

[CL85] Philip R. Cohen and Hector J. Levesque. Speech acts and rationality. In *Proceedings of the Twenty-Third Annual Meeting of the Association for Computational Linguistics*, pages 49–60, University of Chicago, Chicago, Illinois, July 8-12 1985.

[Cla83a] William J. Clancey. The advantages of abstract control knowledge in expert system design. In *Proceedings of the Third National Conference on Artificial Intelligence*, pages 74–78, Washingtion, D.C., August 22-26, 1983.

[Cla83b] William J. Clancey. The epistemology of a rule-based expert system: a framework for explanation. *Artificial Intelligence*, 20(3):215–251, 1983.

[CP79] Philip R. Cohen and C. Raymond Perrault. Elements of a plan-based theory of speech acts. *Cognitive Science*, 3:177–212, 1979.

[CSB84] William J. Clancey, Edward H. Shortliffe, and Bruce G. Buchanan. Intelligent computer-aided instruction for medical diagnosis. In *Readings in Medical Artificial Intelligence: The First Decade*, pages 256–274, Addision-Wesley, 1984.

[Dav76] Randall Davis. *Applications of Meta-level Knowledge to the Construction, Maintenance, and Use of Large Knowledge Bases.* PhD thesis, Stanford University, 1976.

[Gri75] Joseph E. Grimes. *The Thread of Discourse.* Mouton, The Hague, Paris, 1975.

[Gro77] Barbara J. Grosz. *The Representation and Use of Focus in Dialogue Understanding.* Technical Report 151, SRI International, Menlo Park, CA., 1977.

[GS86] Barbara J. Grosz and Candace L. Sidner. Attention, intention, and the structure of discourse. *Computational Linguistics*, 12(3):175–204, 1986.

[GSP78] G.A. Gorry, H. Silverman, and S.G. Pauker. Caputuring clinical expertise: a computer program that considers clinical responses to digitalis. *American Journal of Medicine*, 64:452–460, March 1978.

[HCR84] D. W. Hasling, W. J. Clancey, and G. Rennels. Strategic explanations for a diagnostic consultation system. *International Journal of Man-Machine Studies*, 20(1):3–19, January 1984.

[Hen77] Gary G. Hendrix. *Human Engineering for Applied Natural Language Processing.* Technical Report 139, SRI International, February 1977.

[Hov88] Eduard H. Hovy. Planning coherent multisentential text. In *Proceedings of the Twenty-Sixth Annual Meeting of the Association for Computational Linguistics*, State University of New York, Buffalo, New York, 7-10 June 1988.

[LA87] Diane J. Litman and James F. Allen. A plan recognition model for subdialogues in conversations. *Cognitive Science*, 11:163–200, 1987.

[Lan87] Curtis P. Langlotz. Advice generation in an axiomatically-based expert system. In *Proceedings of the Eleventh Annual Conference on Computer Applications in Medical Care*, pages 49–55, Washingtion, D.C., November 1-4 1987.

[Leh78] Wendy G. Lehnert. *The Process of Question Answering.* Lawrence Erlbaum Associates, Hillsdale, New Jersey, 1978.

[LFT+87] C. P. Langlotz, L. M. Fagan, S. W. Tu, B. I. Sikic, and E. H. Shortliffe. A therapy planning architecture that combines decision theory and artificial intelligence techniques. *Computers and Biomedical Research*, 20(3):279–303, 1987.

[Lit85] Diane Litman. *Plan Recognition and Discourse Analysis: An Integrated Approach for Understanding Dialogues.* PhD thesis, University of Rochester, 1985. Published by University of Rochester as Technical Report TR 170.

[McC85a] Kathleen F. McCoy. *Correcting Object-Related Misconceptions.* PhD thesis, University of Pennsylvania, December 1985. Published by University of Pennsylvania as Technical Report MS-CIS-85-57.

[McC85b] Kathleen F. McCoy. The role of perspective in responding to property misconceptions. In *Proceedings of the Ninth International Joint Conference on Artificial Intelligence*, pages 791–793, IJCAI, Los Angeles, CA, August 1985.

[McK82] Kathleen R. McKeown. *Generating Natural Language Text in Response to Questions About Database Structure.* PhD thesis, University of Pennsylvania, 1982. Published by University of Pennsylvania as Technical Report MS-CIS-82-5.

[McK84] Kathleen R. McKeown. Natural language for expert systems: comparisons with database systems. In *Proceedings of the Tenth International Conference on Computational Linguistics*, pages 794–798, Stanford University, July 1984.

[MM83] William C. Mann and Christian Matthiessen. *Nigel: A Systemic Grammar for Text Generation.* Technical Report RR-83-105, USC/Information Sciences Institute, February 1983.

[Moo89] Johanna D. Moore. Responding to "Huh?": answering vaguely-articulated follow-up questions. In *Proceedings of the Conference on Human Factors in Computing Systems*, Austin, Texas, April 30 - May 4 1989.

[MP88] Johanna D. Moore and Cecile L. Paris. Constructing coherent text using rhetorical relations. In *Proceedings of the Tenth Annual Conference of the Cognitive Science Society*, Montreal, Quebec, August 17-19 1988.

[MS88] Johanna D. Moore and William R. Swartout. A reactive approach to explanation. 17-21 July 1988. Presented at the Fourth International Workshop on Natural Language Generation.

[MT88] William C. Mann and Sandra A. Thompson. Rhetorical structure theory: towards a functional theory of text organization. *TEXT*, 8(3):243–281, 1988.

[MWM85] Kathleen R. McKeown, Myron Wish, and Kevin Matthews. Tailoring explanations for the user. In *Proceedings of the Ninth International Joint Conference on Artificial Intelligence*, pages 794–798, IJCAI, Los Angeles, CA, August 1985.

[NSM85] Robert Neches, William R. Swartout, and Johanna D. Moore. Enhanced maintenance and explanation of expert systems through explicit models of their development. *IEEE Transactions on Software Engineering*, SE-11(11), November 1985.

[Par85] Cecile L. Paris. Description strategies for naive and expert users. In *Proceedings of the Twenty-Third Annual Meeting of the Association for Computational Linguistics*, pages 238–245, University of Chicago, Chicago, Illinois, July 8-12 1985.

[Par87] Cecile L. Paris. *The Use of Explicit User Models in Text Generation: Tailoring to a User's Level of Expertise*. PhD thesis, Columbia University, October 1987.

[PC84] Jasmina Pavlin and Daniel D. Corkill. Selective abstraction of ai system activity. In *Proceedings of the National Conference on Artificial Intelligence*, pages 264–268, Austin, Texas, August 6-10 1984.

[PHW82] Martha E. Pollack, Julia Hirschberg, and Bonnie Lynn Webber. *User Participation in the Reasoning Processes of Expert Systems*. Technical Report CIS-82-10, University of Pennsylvania, 1982. A short version of this report appears in the Proceedings of the Second National Conference on Artificial Intelligence 1982.

[Ric79] Elaine Rich. User modeling via stereotypes. *Cognitive Science*, 3:329–354, 1979.

[Rob84] Jane J. Robinson. *Extending Grammars to New Domains*. Technical Report ISI/RR-83-123, USC/Information Sciences Institute, January 1984.

[Rub85] Robert Rubinoff. *Explaining concepts in Expert Systems: The CLEAR System*. Technical Report MS-CIS-85-06, University of Pennsylvania, 1985.

[Sac75] Earl D. Sacerdoti. *A structure for plans and behavior*. Technical Report TN-109, SRI, 1975.

[Sho76] Edward H. Shortliffe. *Computer Based Medical Consultations: MYCIN*. Elsevier North Holland Inc., 1976.

[SI81] Candace L. Sidner and David Israel. Recognizing intended meaning and speaker's plans. In *Proceedings of the Seventh International Joint Conference on Artificial Intelligence*, pages 203–208, IJCAI, Vancouver, B. C., Canada, August 1981.

[Sid79] Candace L. Sidner. *Toward a Computational Theory of Definite Anaphora Comprehension in English Discourse*. PhD thesis, Massachusetts Institute of Technology, Cambridge, Mass., 1979.

[Spa84] Karen Sparck Jones. *User Models and Expert Systems*. Technical Report No. 61, University of Cambridge Computer Laboratory, December 1984.

[SS87a] William R. Swartout and Stephen W. Smoliar. Explaining the link between causal reasoning and expert behavior. In *Proceedings of the Symposium on Computer Applications in Medical Care*, Washington, D. C., November 1987. (also to appear in "Topics in Medical Artificial Intelligence"; Miller, P.L. (ed), Springer-Verlag).

[SS87b] William R. Swartout and Stephen W. Smoliar. On making expert systems more like experts. *Expert Systems*, 4(3), August 1987.

[Swa77] William R. Swartout. *A Digitalis Therapy Advisor with Explanations*. Technical Report Laboratory for Computer Science TR-176, Massachusetts Institute Technology, February 1977.

[Swa81] William R. Swartout. *Producing Explanations and Justifications of Expert Consulting Systems*. PhD thesis, Massachusetts Institute of Technology, January 1981. Published by Massachusetts Institute of Technlogy as Technical Report Number MIT/LCS/TR-251.

[Swa83] William R. Swartout. XPLAIN: a system for creating and explaining expert consulting systems. *Artificial Intelligence*, 21(3):285–325, September 1983. Also available as ISI/RS-83-4.

[TS84] Randy L. Teach and Edward H. Shortliffe. An analysis of physicians' attitudes. In *Rule-Based Expert Systems: The MYCIN Experiments of the Stanford Heuristic Programming Project*, chapter 34, pages 635–652, Addison-Wesley Publishing Company, 1984.

[Wei80] J. L. Weiner. BLAH, a system which explains its reasoning. *Artificial Intelligence*, 15:19–48, 1980.

[WJ82] Bonnie Lynn Webber and Aravind Joshi. *Taking the Initiative in Natural Language Data Base Interactions: Justifying Why.* Technical Report MS-CIS-82-1, University of Pennsylvania, 1982.

[WK72] W. A. Woods and R. M. Kaplan. *The Lunar Sciences Natural Language Information System: Final Report.* Technical Report, Bolt, Beranek, and Newman, Inc., Cambridge, MA, 1972. BBN Technical Report 2265.

[WS82] J. W. Wallis and E. H. Shortliffe. Explanatory power for medical expert systems: studies in the representation of causal relationships for clinical consultations. *Methods of Information in Medicine*, 21:127–136, 1982.

[WS84] Jerold W. Wallis and Edward H. Shortliffe. Customized explanations using causal knowledge. In *Rule-Based Expert Systems: The MYCIN Experiments of the Stanford Heuristic Programming Project*, chapter 20, pages 371–388, Addison-Wesley Publishing Company, 1984.